

Acadela: A Domain-Specific Language for Modeling Executable Clinical Pathways

Minh Trí Huỳnh

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology
der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften (Dr. rer. nat)

genehmigten Dissertation.

Vorsitz:

Prof. Dr. Alexander Pretschner

Prüfer der Dissertation:

1. Prof. Dr. Florian Matthes
2. Prof. Dr. Martin Bichler

Die Dissertation wurde am 17.08.2023 bei der Technischen Universität München eingereicht und
durch die TUM School of Computation, Information and Technology am 24.01.2024 angenommen.

Abstract

Motivation: E-health applications leverage clinical pathway (CPs) models to execute and manage treatment cases and procedures in medical facilities. Graphical domain-specific languages (DSLs) assist technical experts in modeling and visualizing relevant aspects of CPs, resulting in intuitive, overarching workflow representations. However, expressing graphical notations and medical concepts of CP-specific elements requires intensive extensions to the underlying process modeling frameworks. Meanwhile, text-based DSLs address this concern by textually defining CP concepts in grammar or meta-models. Nevertheless, they can be technical-oriented or without process visualization, limiting their usability and explanatory power.

Methodology: To alleviate these disadvantages, our study develops Acadela, a low-tech-oriented, adaptive text-based DSL with visualization capability to define CPs. Acadela declares grammar to enforce textual syntax for modeling CP concepts, such as treatment procedures, responsibility, medical data visualization, and communications with external systems. The CP concepts conform to the requirements of Smart Adaptive Case Management (SACM), an e-Health system for delivering integrated care for chronic diseases. Furthermore, Acadela provides a visualization to preview the CP model. This feature addresses the limitation of textual DSLs and aims to foster communication between medical and technical staff.

Evaluation: To explore the DSL's expressiveness and usability, we conducted two separate experimental user studies with six medical professionals and eight technical staff. First, we modeled five CPs used by medical professionals in their daily routines. Then through semi-structured interviews, we collected feedback regarding the language's expressiveness. Next, we invited the technical staff to model a hypertension CP and debug a faulty CP model written in Acadela.

Result: Overall, the medical professionals consider the modeled CPs accurately reflect their treatment procedure, and the technical staff consider the language easy to use and applicable to model CPs. The results imply the DSL's potential to model CPs with various degrees of complexity while being learnable and usable to modelers.

Limitation: The participants state that Acadela shall model different data visualization formats, such as infographics, videos, or statistics. Furthermore, the Acadela IDE does not auto-complete the ID of CP elements and preview their UI design. Regarding external validity, Acadela demonstrates that the generated CPs are compatible with SACM, yet the DSL does not experiment with compiling CPs to the format of other e-Health platforms.

Future Work: Acadela has not explored the modeling of CPs in different medical fields, such as Cardiology or Orthopedics. Additionally, the DSL does not model CPs compatible with an e-Health platform other than SACM. Therefore, a potential research direction is to experiment with the challenges and possibilities of modeling other CPs, potentially from different medical fields, used in an existing e-Health platform.

Zusammenfassung

Motivation: E-Health-Systeme nutzen Modelle für klinische Behandlungspfade (KBs), um Fälle und Verfahren in medizinischen Einrichtungen auszuführen und zu verwalten. Grafische domain-spezifische Sprachen (DSLs) unterstützen technische Experten bei der Modellierung und Visualisierung relevanter Aspekte von KBs, was zu intuitiven, übergreifenden Workflow-Darstellungen führt. Allerdings erfordert die Darstellung grafischer Notationen und medizinischer Konzepte von KB-spezifischen Aspekten intensive Erweiterungen der bestehenden Prozessmodellierungs-Frameworks. Textbasierte DSLs adressieren dieses Problem, indem sie KB-Konzepte in Grammatiken oder Metamodellen textuell definieren. Sie können jedoch technisch orientiert oder ohne Prozessvisualisierung sein, was ihre Benutzerfreundlichkeit und Erklärungskraft einschränkt.

Methodik: Um diese Nachteile zu mildern, wird in unserer Studie Acadela entwickelt, eine low-tech-orientierte, adaptive textbasierte DSL mit Visualisierungsfunktion. Acadela deklariert eine Grammatik zur Durchsetzung einer textuellen Syntax für die Modellierung von KB-Konzepten, wie z.B. Behandlungsverfahren, Verantwortung, Visualisierung medizinischer Daten und externe Kommunikation. Die KB-Konzepte entsprechen den Anforderungen von Smart Adaptive Case Management (SACM), einem e-Health-System für die integrierte Versorgung chronischer Krankheiten. Darüber hinaus bietet Acadela eine Visualisierung zur Vorschau des KB-Modells. Diese Funktion adressiert die Beschränkung von textuellen DSLs und zielt darauf ab, die Kommunikation zwischen medizinischem und technischem Experten zu fördern.

Bewertung: Wir führten zwei separate experimentelle Nutzerstudien mit sechs medizinischen Fachkräften und acht technischen Mitarbeitern durch, um die Ausdruckskraft und Nutzbarkeit der DSL zu bewerten. Zunächst modellierten wir fünf KBs, die von den medizinischen Experten in ihrer täglichen Routine verwendet werden. Anschließend sammelten wir in halbstrukturierten Interviews Feedback zur Ausdruckskraft der Sprache. Anschließend luden wir die Technikexperten ein, einen Hypertonie-KB zu modellieren und ein fehlerhaftes KB-Modell zu debuggen.

Ergebnis: Die medizinischen Fachkräfte finden, dass die modellierten KB ihr Behandlungsverfahren genau widerspiegeln, und das Technikpersonal betrachtet die Sprache als nutzbar und geeignet für die KB-Modellierung. Die Ergebnisse zeigen, dass die DSL KBs mit verschiedener Komplexität modellieren kann und dabei für Modellierer erlernbar und nutzbar ist.

Einschränkung: Die Teilnehmer geben an, dass Acadela verschiedene Datenvisualisierungsformate modellieren soll, wie z.B. Infografiken, Videos oder Statistiken. Außerdem vervollständigt die Acadela IDE nicht automatisch die ID von KB-Elementen und zeigt deren UI-Design in der Vorschau an. Was die externe Validität betrifft, so zeigt Acadela, dass die generierten KBs mit SACM kompatibel sind, doch experimentiert die DSL nicht mit der Kompilierung von KBs in das Format anderer e-Health-Plattformen.

Zukünftige Arbeit: Acadela sollte die Modellierung von KBs in verschiedenen medizinischen Bereichen, wie Kardiologie oder Orthopädie, demonstrieren. Außerdem sollte die DSL die Kompatibilität mit anderen e-Health-Plattformen als der SACM zeigen. Auf diese Weise kann man die Herausforderungen und Möglichkeiten der Modellierung und Ausführung von KBs aus verschiedenen medizinischen Bereichen in einem e-Health-System entdecken.

Acknowledgements

I would like to express my gratitude to Professor Florian Matthes, who has been offering constant support throughout my Doctoral study with great patience, generosity, and valuable guidance. The experience, knowledge, and expectation that I learn from Professor Matthes are essential not only for a broad and in-depth study of my field of research but also for managerial aspects relevant to an efficient organization of research or teaching activities in my current and future career. In addition, I highly appreciate one research ideology of Professor Matthes that technology or research shall not only bridge the existing gap or expand the current knowledge base, but it shall deliver practical benefits and address the concerns of its beneficiaries. Therefore, I have been endeavoring to apply this inspiration in my research work and teaching during my Doctorate in the chair of Software Engineering for Business Information Systems (sebis).

My Masters and Doctorate would not exist without the help and encouragement of my mother. Therefore, I would like to express my gratefulness for her immeasurable patience, sympathy, and suffering to spiritually and financially support me since my birthday. May the accomplishment of this Dissertation partly repay the unconditional sacrifice and magnificent merit of my mother.

My Thesis leverages the support of Smart Adaptive Case Management (SACM), textX, and GoJS to deliver its functionality. Hence I highly appreciate the support and patience of Felix Michel, who enlightened me on the mechanism of SACM. Furthermore, I also wish to thank Igor Dejanović and the GoJS development team for creating user-friendly and maintainable technical solutions that significantly support me in realizing the features of my Dissertation.

Without the valuable feedback of the participants, I would not be able to evaluate the applicability and quality of the research. Thus, I sincerely thank the medical professionals and technical staff, who devoted a remarkable proportion of their hectic schedules to help me understand the practical value which I can contribute to their work and the healthcare domain.

Working at the sebis chair of Technical University Munich significantly extends my knowledge and experience in research, teaching, and interpersonal collaboration. Therefore, I appreciate my colleagues and all other staff at TUM for their valuable support, openness, understanding, and advice that greatly expand my limited horizon, develop cultural and emotional awareness, and assist me to focus on accomplishing my duty and research.

Finally, I would like to express my appreciation to my wife, Thoang Thanh Mai, for her empathy, encouragement, and support in terms of spirit and nutriment. Her motivation and companionship significantly power me to accelerate and overcome barriers in accomplishing my Thesis work. Furthermore, I also would like to thank my sister Mai Huynh, for her spiritual support and cheering throughout my Doctoral study.

Unterschleißheim, Bavaria, Germany, 22.01.2023
Tri Huynh

Contents

Abstract	iii
Zusammenfassung	iv
Acknowledgements	v
Contents	vi
List of Figures	x
List of Tables	xvi
1 Introduction	1
1.1 Problem Description	1
1.2 Research Questions	2
1.3 User Study	4
1.4 Dissertation Structure	5
2 Foundations	6
2.1 Clinical Pathways (CP)	6
2.1.1 Goals	6
2.1.2 Characteristics	7
2.2 Adaptive Case Management (ACM)	9
2.2.1 Concerns of Production Case Management	9
2.2.2 Suitability for Clinical Pathways Modeling	10
2.2.3 Challenges	12
2.3 Case Management Model and Notation (CMMN)	12
2.3.1 Applicability to ACM and CP Modeling	13
2.3.2 Concepts and Graphical Notations	14
2.4 Domain Specific Language (DSL)	19
2.4.1 Modeling Language Elements	19
2.4.2 Advantages of Domain Specific Modeling Languages	19
2.4.3 Graphical Domain Specific Languages	21
2.4.4 Textual Domain Specific Languages	22
2.5 textX Meta-Language (Compiler-compiler)	23
2.5.1 Motivation	23
2.5.2 Features	24
3 Related Work	27
3.1 Graphical Domain-Specific Modeling Languages	27
3.1.1 DSML4CPs - Modeling Clinical Pathways in Oncology Using Extended MEMO OrgML Process Modeling Language	27
3.1.2 BPMN4CP - Modeling Clinical Pathways by Extending BPMN	29

CONTENTS

3.1.3	BPMN ^{SIX} - Modeling Surgical Workflow by Extending BPMN	31
3.2	Textual Domain-Specific Language	32
3.2.1	FCIG - Modeling Clinical Guidelines using Xtext	32
3.2.2	Prescriptive Grammar for Clinical Describing Workflow	33
4	Smart Adaptive Case Management (SACM)	35
4.1	Problem Description	35
4.2	Requirements	37
4.2.1	R1: Support a Purely Meta-Model-Based Approach	37
4.2.2	R2. Integration with External Services	40
4.2.3	R3: Support Communication and Coordination	41
4.3	Architecture	43
4.3.1	Conceptual Layers of SACM Backend	43
4.3.2	The CONNECARE Project	46
4.3.3	SACM-CONNECARE Integration	47
4.4	Meta-model Elements in SACM	50
4.4.1	■ Schemata and ■ Data	51
4.4.2	■ Actors	51
4.4.3	■ Case Definition	52
4.4.4	■ User Interface	53
4.4.5	■ Case	53
5	Language Design	55
5.1	Requirements for Modeling Clinical Pathways	55
5.2	Language Specification	60
5.2.1	Flexible Syntactic Rules	60
5.2.2	Automatic Execution of Default Behaviors	62
5.2.3	Concise Constructs	63
5.3	Concrete Syntax for Clinical Pathway Element Definition	66
5.3.1	Data Type	66
5.3.2	Mandatory Attribute	68
5.3.3	Input Field	68
5.3.4	Output Field	71
5.3.5	Form	76
5.3.6	Precondition	76
5.3.7	Trigger (HttpHook)	78
5.3.8	Task	80
5.3.9	Stage	85
5.3.10	Summary Panel	88
5.3.11	Responsibilities	91
5.3.12	Setting	92
5.3.13	Case	93
5.3.14	Workspace	96
5.3.15	Import	96
5.4	Constraint Validation	97
5.4.1	Syntax Errors	98
5.4.2	Semantic Errors	101
5.5	Syntax Optimization Effect	105

CONTENTS

6	Implementation	107
6.1	Architecture Design	107
6.1.1	Acadela System Components	108
6.2	Integrated Development Environment (IDE)	110
6.3	Grammar Definition	112
6.3.1	textX Grammar Rule Expressions	113
6.3.2	Grammar Specification for Modeling Clinical Pathways	116
6.3.3	Reflection on Addressing CP Modeling Requirements	125
6.4	Parser	126
6.5	Interpreter	128
6.5.1	CP Meta-model Construction	129
6.5.2	Syntax Error Validation	133
6.5.3	Semantic Error Validation	135
6.6	Compilation to SACM Clinical Pathway	138
6.7	Model Visualization	149
6.7.1	Graphical Notation Definition	149
6.7.2	Rendering CP Elements	150
6.7.3	Double-clicking to Focus on the Code Definition	153
7	Evaluation	154
7.1	Evaluation Approach	154
7.1.1	Define Evaluation Goals and Scopes	154
7.1.2	Identify Units of Analysis	155
7.1.3	Design Evaluation Tasks	155
7.1.4	User Study Setup	156
7.1.5	Pilot Testing	156
7.1.6	Schedule User Study	156
7.1.7	Data Collection	157
7.1.8	Draw Individual and Collective Results	157
7.1.9	Identify Implications	157
7.2	Expressiveness	157
7.2.1	Population	158
7.2.2	Modeled Clinical Pathways	158
7.2.3	User Study Setup	162
7.2.4	Result	162
7.2.5	Discussion	164
7.3	Usability Evaluation	165
7.3.1	Population	165
7.3.2	Experiment Setup	165
7.3.3	Experiment Design	165
7.3.4	Training	165
7.3.5	Modeling	166
7.3.6	Result	167
7.3.7	Discussion	169
7.4	Limitations	170
7.4.1	Need Supportive CP Elements	170
7.4.2	Support Previewing CP Elements	170
7.4.3	Auto-complete CP Elements in Web-based IDE	170
7.4.4	Dependent on textX Error Handler	170

CONTENTS

7.4.5	Mishandling of Special Characters	170
7.4.6	Limited Number of Participants	170
7.4.7	Internal Validity	171
7.4.8	External Validity	171
8	Conclusion and Future Work	172
8.1	Summary	172
8.2	Future Work	175
	References	175
A	Appendices	184
A.1	Acadela Complete Grammar	184
A.2	CP Model Definition and Visualization in Acadela	198
A.2.1	COPD Breathing Exercise	198
A.2.2	Selection of Antipsychotics for Schizophrenia	202
A.2.3	Diagnosis of Class II Smoke Inhalation Injury	208
A.2.4	Cervical Cancer Diagnosis	214
A.2.5	Chronic Headache Treatment	220
A.3	Code Snippets	228
A.3.1	Responsibilities Declaration and Assignment	228
A.3.2	Dynamic Template Definition	228
A.4	System Usability Scale Questionnaire	230
A.5	Syntax and Semantic Error Analyzer	230
A.5.1	Mapping of Violated Rules and their Human-readable Representation	230
A.6	HttpHook Example Content	231
A.7	Complete SACM Meta-model	234

List of Figures

2.1	ACM Key Concepts.	10
2.2	Design-time phase modeling and runtime phase planning	13
2.3	Graphical Notations of four Task types expressed in each column with the bottom row shows their Discretionary version. a1) Non-blocking HumanTask, a2) Blocking HumanTask, b) ProcessTask, c) CaseTask, d) DecisionTask.	14
2.4	Graphical Notations of a CMMN CaseFileItem.	15
2.5	Graphical Notations of TimerEventListener (left) and UserEventListener (right). . .	15
2.6	CMMN Sentry establishes a dependency between two Tasks (left) or two Stages (right). The white diamond represents an Entry Criterion, while the black diamond denotes an Exit Criterion.	16
2.7	CMMN depicts an AND (left) or OR (right) relationship among Tasks using Sentry as Entry Criteria.	16
2.8	A Milestone (left) with one Entry Criterion (right) notations in CMMN.	16
2.9	Graphical Notations of a CMMN Stage (left) and Discretionary Stage(right). . .	17
2.10	CMMN Graphical Notation of a CasePlanModel. The folder body includes all Case elements	17
2.11	CMMN Applicable Decorator to CaseItems (Object Management Group, 2016, p. 79).	18
2.12	Illustration of a DSML mechanism that identifies domain-specific concepts from the business domain and represents the concepts through an implementation language (DSL). (Frank, 2011 a, p. 29).	20
2.13	Simplified textX architecture and workflow from a) creating <i>parser</i> and <i>meta-model</i> from grammar, b) parsing the model definition code to generate a model, and c) interpreting or generating code. (Dejanović, Vaderna, Milosavljević, & Vuković, 2017, p. 3).	24
2.14	textX example workflow for processing a Robot Instruction Language. a) language grammar specification, b) <i>meta-model</i> generated from the grammar, c) the source code of a Robot Instruction set, d) the generated model (Dejanović, n.d.e).	25
2.15	Arpeggio ParsingExpression class and example. a) The hierarchy of PEG's classes in Arpeggio, b) The <i>parser</i> model of the Robot Instruction Language (Dejanović, Milosavljević, & Vaderna, 2016, p. 3).	25
2.16	A Python program to interpret the Robot Instruction model using textX facilities.	26
3.1	Excerpt of a Soft Tissue Sarcoma CP expressed in DSML4CPs (Heß, Kaczmarek, Frank, Podleska, & Täger, 2015, p. 12).	28
3.2	BPMN4CP model of a (simplified) stroke CP with workflow, resources and documents (Braun, Schlieter, Burwitz, & Esswein, 2016, p. 29).	29
3.3	BPMN ^{SIX} model of a cataract surgery process and a part of <i>Phacoemulsification</i> subprocess	31
3.4	A CIG model (left) and its outline (right) in FCIG IDE	33
3.5	A prescription workflow that the DSL models.	34

LIST OF FIGURES

4.1	Visualization of the problem in the Integrated Care Environment without integrated tool support (left). Each organization stores medical data independently, leading to redundancy and uncoordinated analysis of the patient’s medical status, potentially resulting in undesirable outcomes. An integrated care environment empowered by integrated tools (right) can offer consistent documentation of critical data, thus fostering collaboration and communication among care professionals throughout the care process. (Michel, 2020, p. 3)	36
4.2	Degree of process structure according to Michel et al. (adapted from (Di Ciccio, Marrella, & Russo, 2012))	39
4.3	Conceptual architectural layers of SACM (Hernandez-Mendez, Michel, & Matthes, 2018, p. 263)	44
4.4	Capabilities ordered by conceptual layers. Solid lines represent the usage of a specific capability. Dashed lines represent extended functionality. Adapted from Hernandez-Mendez et al. (2018, p. 265) by Michel (2020).	45
4.5	High-level project vision. The Smart Adaptive Case Management provides care services with its collaborative, purely meta-model-based approach enriched with clinical decision support (Michel, 2020, p.135).	47
4.6	The CONNECARE system architecture adapted from (Michel & Matthes, 2018, p. 17). ACM4IC Components are highlighted with a dot in the upper right corner (Michel, 2020, p. 141)	48
4.7	Conceptual orchestration of a monitoring prescription task (Michel, 2020, p. 143). 50	50
4.8	Meta-model adapted from Hernandez-Mendez et al. (2018, p. 267). A complete conceptual meta-model containing relevant attributes is illustrated in Figures A.6, A.7, and A.8 of Appendix A.7	50
5.1	Illustration of Flexible Syntax Rules in Acadela. The two code snippets are syntactically different but semantically equivalent. The comparison demonstrates the Acadela properties of 1) Case-insensitive, 2) Indent-insensitive, 3) No Announcement Separator for each key-value assignment, 4) No End Indicator for each CP element declaration, 5) Interchangeable Quote and Double Quote for String, 6) Directives to represent system-defined attribute values, 7) Interchangeable Order of Directives, Child Elements, and Attributes	61
5.2	Illustration of default constant value assignment in Acadela. In SACM (1), modelers need to declare the <i>mandatory</i> Attribute whether the Stage is <i>mandatory</i> or not. In Acadela (2), <i>mandatory</i> is the default value of the <i>mandatory</i> Attribute, and <i>repeatable</i> has <i>ONCE</i> as the default value, hence modelers do not need to specify it.	63
5.3	Illustration of Acadela approach to combine <i>schema</i> , <i>execution behaviors</i> and <i>visual representation</i> in one single CP element definition. The HumanTask definition in SACM (1) requires the schema definition (EntityDefinition) and case-related execution and visualization (HumanTaskDefinition). Meanwhile Acadela (2) groups all HumanTask attributes, behaviors, and graphical representation into a single element.	64
5.4	Hierarchy of elements in a clinical pathway model expressed in Acadela.	65
5.5	Input Field Rendering in SACM produced by Listing 5.1	70
5.6	Rendering of the Multiple Choice - Single Answer InputField in the SACM UI. The InputField value in this example is '1'.	71
5.7	Rendering of the Multiple Choice - Multiple Answer InputField in the SACM UI. The InputField value in this example is ['TEMPLE', 'SHOULDER'].	71

LIST OF FIGURES

5.8	SACM Display of the Conditional OutputField in Listing 5.5. Here the blood pressure is high because the Diastolic value is above 90.	74
5.9	Background Color Effect from the color code definition to the BmiValue Output-Field in Listing 5.6.	74
5.10	Illustration of InputField values (massage positions - left) affect the output image visualization (right) by showing temple and nape massage positions as blue circles.	75
5.11	Acadela code (left) to declaring conjunction (AND) relationship between prerequisite Stages or Tasks in Acadela. The SACM UIs (right) display the <i>Assessment</i> Stage is activated only when the <i>Cardiogram</i> and <i>MRI Scan</i> Stages finish (Picture c). If both or one of the <i>Cardiogram</i> or <i>MRI Scan</i> Stages are activated (Picture a and b), the <i>Assessment</i> Stage is disabled.	78
5.12	Acadela Declaration (left) and SACM UI (right) of a single Task (Quick Test) and multiple instances of a repeatable Task (PCR Test) for the <i>Lab Test</i> Stage. An Add Task button exists when there is at least one manually activated Task in the Stage. Clicking the Add Task button allows the creation of the repeatable Task(s), as shown in the below dropdown box. The example inspires by the research on applying parallel PCR Tests to detect SARS-CoV-2 by (Perchetti et al., 2020, p. 2)	83
5.13	Acadela Declaration (left) and SACM UI (right) of a single Stage (Identification) and parallelly repeatable Stage (PCR Test). The circle with a plus icon exists when declaring at least one manually activable Stage. Clicking the plus circle displays a dropdown box to select the (repeatable) Stage, which SACM will instantiate. The number below the parallel Stage is the cardinal instance number, not the latest iteration of that Stage. Each parallel Stage is executable only once. The example inspires by the research on applying parallel PCR Tests to detect SARS-CoV-2 by (Perchetti et al., 2020, p. 2)	87
5.14	UI display of a repetitive Stage in SACM. A blank circle expresses a completed Stage. The door-shaped purple object denotes the activated <i>Exercise</i> Stage that the user is interacting with. SACM shows the latest Stage iteration below its name.	88
5.15	Illustration of SummaryPanel UI in SACM as defined in Listing 5.18. SACM retains the visual effect of each Input/OutputField.	90
5.16	UI of the Case defined in Listing 5.23 from the view of a Clinician User. The screenshot shows the current workflow with completed and opening Stages (purple-background circles), Tasks, assigned roles, and other Case information (e.g., Case name, patient info).	96
5.17	Example of invalid element declaration (top) and their enhanced error message (bottom) in Acadela. The Form <i>UnexpectedForm</i> is invalid as Stages do not accept a Form as a child element.	98
5.18	Example of the need for customizing the textX error message. Original error message (a) of textX that shows the grammar construct ('Eq') in stead of the concrete syntax element ("="). Therefore, Acadela enhances the message (b) to show the expected operator.	99
5.19	Example of an error message (right) for typos that are similar to Acadela keyword (left). At line 97, the typo "Precondition" has two different characters than the "Precondition" keyword of Acadela. Thus the error message suggests the correct keyword to modelers.	99

LIST OF FIGURES

5.20 Example of an error message (right) for typos that are totally different from Acadela keywords (left). In line 97, the typo "Prerequisite" does not exist in the Acadela dictionary, and many characters are not closely similar to any Acadela keyword. Thus Acadela only states that the keyword is unrecognized. 100

5.21 Example of an enhanced error message (b) in Acadela from the original one of textX (a) to explain the invalid String data type (top). In line 133, a single quote is missing in the String value. Thus Acadela informs modelers that quotation marks are needed. 100

5.22 Example of customized syntax validation for conditional statements in the expression attribute. The top case shows an error of using the wrong keyword (":" instead of "then"); the bottom one demonstrates an incomplete boolean expression. Acadela outputs the corresponding enhanced error messages below the code snippet to state the error cause and solution direction. 101

5.23 Example of invalid referencing path in the condition attribute of a Precondition. At line 12, the *RequestMedicalLabTest* Task ID of the path does not refer to an existing Stage. The intended Task ID is *MeasureBloodPressure*. Finally, Acadela gives a hint that it expects an existing Task ID. 102

5.24 Example of invalid referenced ID in the *previousStep* attribute of a Precondition. At line 11, the *CholesterolTest* ID refers to an InputField but not an existing Stage or Task. The intended ID is *RequestMedicalTest*. Finally, Acadela states that an existing ID of a Stage or Task is required. 102

5.25 Example of invalid referenced ID in the expression attribute of an OutputField. At line 10, 11, and 12, the *SystolicValue* ID does not refer any InputField or OutputField in the same Form. The intended ID is *Systolic*. Therefore, Acadela suggests to declare an InputField or OutputField within the same Form of the OutputField. 103

5.26 Example of an enhanced error message for duplicate Stage IDs. 103

5.27 Example of an enhanced error message for duplicate Stage and Task IDs. 104

5.28 Example of an enhanced error message for sending requests to untrusted URLs of external services. 104

5.29 Example of an enhanced error message for sending requests to trusted URLs but using unauthorized HTTP Method. In this example, the trusted URL does not allow SACM to use the DELETE method. 104

5.30 Example definition of a minimum Stage in SACM (left - 279 characters) and Acadela (right - 49 characters) 105

5.31 Example definition of a HumanTask in SACM (left - 2007 characters), and Acadela (right - 891 characters) 106

6.1 Container diagram of the SACM-Acadela system integration. The green containers denote the frontend-related applications, while the blue ones represent the backend applications. 107

6.2 Component diagram of the Acadela frontend. 108

6.3 Component diagram of the Acadela backend. The dark blue components represent classes and libraries used in the Acadela backend. The light blue rounded rectangles are SACM Containers that the Acadela backend communicates with, and the green container is the Acadela frontend React application. 109

6.4 Demonstration of Autocomplete in Acadela IDE. The left picture shows an incomplete *keyword* and *suggestions*. The middle one shows the inserted code snippet after selecting the auto-complete option. Finally, the right picture shows the definition of a **Snippet** auto-complete definition in Monaco. 110

LIST OF FIGURES

6.5	The IDE GUI of Acadela with <i>syntax highlighting</i> . The top panel shows the code written by modelers. The Validate button sends the code to the backend for verifying syntactic or semantic errors. The Submit button has the same feature as the Validate one, but the backend further checks the existence of defined Users and Groups in the CP and sends the CP meta-model to SACM. Finally, the bottom panel shows the status of the code compilation or an enhanced error message if a bug occurs.	111
6.6	Syntax highlighting definition for <i>keywords</i> and <i>code sections</i> in Monaco (bottom) based on a set of color rules (top). Lines 27 and 28 instructs Monaco to identify multi-line strings in the code.	112
6.7	The Acadela CP meta-model based on the SACM schema and Acadela grammar. The default multiplicity is one, e.g., One Case has only one Setting.	130
6.8	Example of how the Acadela Interpreter traverses through attributes in a Stage object of the textX model (top left) to create an Acadela Stage object with attributes and objects of child elements (HumanTasks, Precondition, and HttpHook) on the right. 1) and 2): Extract the Stage attributes from the corresponding textX model object, such as <i>description</i> (1) and <i>ownerPath</i> (2). 3) Construct a Stage object from the extracted attribute values. 4) Access each Task in the Stage to collect Task’s properties. 5a) Construct a Task Object from the Task properties identified in Step 4. 5b) Construct other Tasks in the Stage by repeating Step 4. 6) Extract the previous Stage or Task ID and condition in the Precondition textX model object. 7) Construct the Precondition object based on the attributes extracted in Step 6. 8) For imported elements, e.g., HttpHook, access the reference (ref) attribute to 9) extract the attributes of the imported element. Finally, 10) Construct the HttpHook from the extracted attributes in Step 9.	131
6.9	Example of compiling properties of an interpreted Case object (left) into SACM JSON format (right).	142
6.10	Example of compiling an interpreted SummarySection object (left) into SACM JSON format (right). The SummarySection code snippet is from lines 42 to 49 of Listing 5.23.	143
6.11	Example of compiling properties of an interpreted HumanTask object (left) into SACM JSON format (right). The HumanTask is activated when the Dietician asks for a BMI check in the ExaminationChecklist Task of the same Evaluation Stage.	144
6.12	Example of compiling properties of an interpreted Stage object (left) into SACM JSON format (right).	145
6.13	Example of compiling a interpreted Stage (left) into SACM JSON structure of EntityDefinition (right). The Attribute-related properties of HumanTasks in the Stage constitute their AttributeDefinitions.	146
6.14	Example of compiling a interpreted HumanTask (left) into SACM JSON structure of EntityDefinition (right). The Attribute-related properties of Input/OutputField in the Stage form their AttributeDefinitions.	148
6.15	Example of a CP Visualization in Acadela using the GoJS tool.	149
7.1	Suggestions to conduct user study in research from Robert K. Yin (top) and activities in designing Acadela user study (bottom).	154
7.2	Execution procedure of the breathing exercise in a web application. a) The patient answers questions about health status prior to the exercise. b) A post-questionnaire to record the patient’s condition after the exercise (Cavusoglu, 2021).	159

LIST OF FIGURES

7.3	Forest plots present the effect sizes on efficacy, fatigue, and weight gain for comparisons of antipsychotics and placebo. Relative risks are the measurement unit for effect size, except for weight gain which uses mean differences in kilogram (Siafis et al., 2022).	159
7.4	CP of Smoke Inhalation Injury Assessment. The process is constraint-driven based on the symptoms (Karpov, 2018).	160
7.5	CP of Cervical Cancer Screening. The guideline suggests different treatment processes depending on the patient’s age (Roche Diagnostics, 2018).	161
7.6	Illustration of InputField values (massage positions - left) affect the output image visualization (right)	162
7.7	Rating of medical professionals on Statements regarding the accuracy of modeled CP elements (stage, task, and transition condition), visualization, and treatment process.	163
7.8	Excerpts of feedback towards the modeled CPs from medical professionals. . .	163
7.9	The workflow of a simplified Hypertension CP	167
7.10	Rating of experts on Statements regarding the a) Usability of the Acadela Syntax, b) Usability of EMs, c) Usefulness of EMs, and d) Acadela SUS Score.	168
7.11	Grade Ranking of SUS Score.	168
A.1	Visualization of the CP model generated by Listing A.2.	198
A.2	Visualization of the CP model generated by Listing A.3.	202
A.3	Visualization of the CP model generated by Listing A.4.	208
A.4	Visualization of the CP model generated by Listing A.5.	214
A.5	Visualization of the CP model generated by Listing A.6.	220
A.6	Detailed meta-model with focus on the case definition (Michel, 2020, p.190). .	234
A.7	Detailed meta-model with focus on the schemata and data (Michel, 2020, p.191). .	235
A.8	Detailed meta-model with focus on the case. (Michel, 2020, p.192).	236

List of Tables

2.1	Characteristics of CPs derived from the three articles	8
4.1	High-level responsibilities of components in the CONNECARE system (Michel, 2020, p. 142)	49
5.1	Features Comparison of DSLs for CP modeling.	55
5.2	Description of Data Types in Acadela based on SACM specification. (Michel, 2020, p. 101)	67
5.3	An example of expressing the <i>isMandatory</i> Attribute of a Stage in SACM and Acadela.	68
5.4	The attributes of an Acadela InputField. * - the attribute is required. A - the attribute is created in Acadela and does not exist in SACM.	70
5.5	The attribute of an Acadela OutputField. * - the attribute is required. A - the attribute is created in Acadela and does not exist in SACM.	73
5.6	The Acadela Form attributes apply to all the included InputFields and OutputFields.	76
5.7	The attributes of Acadela Precondition. * - the attribute is required.	77
5.8	Description on the attributes of a HttpHook (Michel, 2020, p. 113). - the attribute is required.	* 79
5.9	The attributes of an Acadela Stage. * - the attribute is required. A - the attribute is created in Acadela and does not exist in SACM.	82
5.10	The attributes of an Acadela Task. * - the attribute is required. A - the attribute is created in Acadela and does not exist in SACM.	82
5.11	The attributes of an Acadela SummaryPanel. * - the attribute is required.	89
5.12	The attributes of an Acadela User. * - the attribute is required.	91
5.13	The attributes of an Acadela Group. * - the attribute is required.	91
5.14	The attributes of an Acadela Setting Attribute. * - the attribute is required.	92
5.15	The attributes of an Acadela Case. * - the attribute is required. A - the attribute is created in Acadela and does not exist in SACM.	94
6.1	Example of a trusted API table as a CSV file. Each row stores the Workspace ID, the API URL of an external system, and its eligible HTTP method(s).	138
7.1	Background of medical experts participated in the expressiveness evaluation.	158
7.2	Background of technical staff participated in the usability evaluation	166
A.1	SUS Statements to Rate Acadela Syntax and Error Validator.	230

1 Introduction

1.1 Problem Description

E-Health applications have been offering essential support to execute, monitor, analyze, and manage treatments and clinical data in medical facilities. Clinical pathways (CPs) serve as the core of e-Health applications as they establish a standardized procedure for medical treatments of a specific patient group in a defined period (Panella et al., 2003; Campbell et al., 1998; Every et al., 2000). Besides regulating workflow activities, CPs also govern the utilization of resources (human, medical, administrative (Heß et al., 2015; Braun et al., 2014)) necessary to provide healthcare services (Khodambashi, 2013). By playing a decisive role in executing medical treatments, CPs significantly influence the quality of healthcare services (Vanhaecht et al., 2009; Rotter et al., 2010; Hai et al., 2019). Effectively applying CPs results in numerous benefits, such as ameliorating patient conditions (Rotter et al., 2010; Panella et al., 2008), reducing complications (Preston et al., 2013), mortality (Hai et al., 2019), length of stay, and treatment costs (Yang & Su, 2014), enhancing communication between medical professionals (Vanhaecht et al., 2009, 2010), and increasing satisfaction for patients (Van Dam et al., 2013) and medical staff (Schuld et al., 2011). For this reason, modeling CPs can improve the quality of healthcare services by enabling process analysis (Fernández-Llatas et al., 2010), variance analysis (Di Lenarda et al., 2017; García et al., 2016; Du et al., 2020), audit (Fudholi & Mutawalli, 2018), or automating process execution (Tongchuan & Deyu, 2013; Li et al., 2014, p. 402).

To realize the benefits of CP models, e-Health systems can apply *Enterprise Modeling* to construct conceptual models for analyzing, describing, and planning organizational aspects (Frank, 2014, p. 2; Sandkuhl et al., 2014, pp. 16-17). Building conceptual models requires a modeling language that comprises syntax and semantics elements to define modeling concepts (Siau & Rossi, 2011, p. 251). Specifically, according to Frank (2011a, pp. 26-27), "*abstract syntax* defines rules for constructing *syntactically correct models* using the language concepts. The *concrete syntax* defines the *symbols* used to represent the abstract syntax. Since these *symbols* are usually graphical, it is also referred to as *graphical notation*. The *semantics* of a modelling language defines the (formal) *interpretation* of modelling concepts." To represent conceptual models, one can apply General Purpose Modeling Language (GPML) or a Domain-Specific (Modeling) Language (DS(M)L). Because the building blocks of DSLs are terminologies and concepts used in the respective domain (Heß et al., 2015), they offer several benefits over GPMLs (Frank, 2013, pp. 133-134).

Firstly, DSML enhances productivity because users do not construct domain concepts on their own (Frank, 2013, p. 133) using primitive data types of GMPL (Heß et al., 2015, p. 3). The second benefit is the consolidation of model integrity, as syntax and semantic rules prevent illogical or illegitimate model definitions to a certain degree (Frank, 2013, p. 133). Consequently, a thoroughly designed language increases the model quality. Finally, DSMLs leverage concrete syntax (e.g., graphical notations) to enhance the comprehensibility of the model (Frank, 2013, p. 134). In combination with *domain-specific technical terms*, DSMLs also *foster communication*

between users of the models (Heß et al., 2015, p. 3). For these reasons, designing effective and efficient DSMLs is one of the state-of-the-art methods for modeling CPs.

One of the approaches to classify DSMLs is by considering the language representation, which leverages graphical notations or text. On the one hand, graphical DSLs enable modelers to build medical procedures by interacting with visual elements in a GUI. One advantage of this approach is that using graphical notations improves the extension and convenience of the model (Frank, 2010, p. 1). If graphical DSLs appropriately display model elements following a logical and hierarchical structure (Wienands & Golm, 2009, p. 458) with domain-specific technical terms, they can provide a user-friendly interface and learnable modeling mechanism (Hermans et al., 2009, p. 433). Furthermore, visual artifacts foster communication between technical and domain experts, as they share a common understanding of the notations for CP elements (Heß et al., 2015). However, one noticeable trade-off is that existing workflow modeling notations, like BPMN, lack the concepts to express aspects besides workflow execution (e.g., resource consumption or documents), thus demanding extra effort to develop custom extensions for modeling and visualizing these aspects (Braun et al., 2014; Heß et al., 2015; Neumann et al., 2016).

On the other hand, textual DSLs model CPs using a text-based interface. The textual presentation of the model combined with an IDE brings three significant benefits. First, extensions of the model are convenient with textual definitions of sub-DSL for new elements (Rieger et al., 2018). Second, validating models is manageable as textual DSLs only define syntactic constraints via *grammar* (Baar, 2015). Last but not least, an IDE offers convenient support with error warning, syntax highlighting, and auto-completion of CP elements and their values (Cook et al., 2007, p. 16-17; Merkle, 2010). However, textual DSLs typically do not offer model visualization, which may hinder users from previewing the modeled process. Furthermore, DSLs shall accommodate several barriers to be practical and user-friendly in modeling CPs.

The first barrier is that DSLs specialized in particular medical fields or treatments due to their focus on specific, intensive medical procedures used in a hospital (Heß et al., 2015). In these cases, extending the DSL to support treatments in other medical fields requires a considerable effort to include new terminology or elements. The second concern is usability, as DSLs should be easy to use and learn for **modelers**. Consequently, DSLs can indirectly the quality of the healthcare service by improving user satisfaction and productivity of the modeling process.

1.2 Research Questions

Our study aims to address the above concerns of DSLs for **modeling CPs used in different medical departments** by proposing a *generic, textual DSL* named Acadela. Additionally, our DSL strives for a *straightforward, flexible, and concise syntax* to **increase** the **learnability** and **usability** on the side of **modelers**. Given the two motivations, we formulate the following hypothesis to discover the feasibility of developing such a DSL:

Hypothesis

It is possible to define a single DSL which can **model** and **orchestrate** CPs while **fostering communication** between clinical and technical experts.

To answer the above hypothesis, our study starts from the foundation of CP modeling by finding the necessary elements to construct executable CPs. Next, we explore how to define the syntax and semantics of our DSL such that it can express diverse CPs while being user-friendly

and learnable. Finally, our study shall evaluate the expressiveness and usability of the DSL by conducting user studies with medical professionals and technical staff at healthcare institutions. Based on this direction, our research formulates the following research questions (RQs) to collect the data and evidence:

Research Question 1

What elements are required to **model** and **orchestrate** executable Clinical Pathways for Adaptive Case Management?

CP modeling shall consider not only elements relevant to workflow execution but also the resources necessary to provide healthcare services. Examples of these resources are involved staff, financial documents, or clinical instructions in textual or video format (Heß et al., 2015). Furthermore, due to the unpredictable character of CPs, the DSL shall be capable of modeling both non-deterministic and structured processes (Michel, 2020, p. 34). Therefore, our study first identifies the essential elements to model CPs through literature research and a study of an existing e-Health system. In addition, the DSL visualizes the modeled CP to provide an overarching overview of the treatment process to modelers and care professionals.

Research Question 2

What are the syntax and semantics that enable a textual DSL to model executable Clinical Pathways for Adaptive Case Management?

The next concern is how to design a DSL to model the identified elements. In other words, how can one formulate the **syntax** and **semantics** to define CP elements in textual format. First, the language representation shall express the *features of a CP model* while being *intuitive* and *learnable* to *modelers*. Therefore, Acadela aims for a low-technical-oriented and concise *syntax* to express CP elements. The second concern is how to define an *abstract syntax* to enforce rules for flexibly presenting the *concrete syntax*. Finally, we discover how to interpret and validate the CP model *semantically*. This step is crucial for compiling a CP model written in Acadela to a format compatible with the experimented e-Health system.

Research Question 3

Can the DSL model Clinical Pathways from **different medical fields** with **diverse complexity** while being **understandable** to **clinical experts**?

This RQ explores 1) the *expressiveness* of the DSL and 2) its *understandability* from the view of **medical professionals**. Since Acadela aims to model generic CPs, our study examines the DSL's capability in modeling *linear* to *complex conditional-driven* CPs in the daily routines of care professionals from various medical departments. Specifically, we modeled the *workflow elements* and *stage transition conditions* for each CP. Then, we interview the experts to evaluate the CP models' accuracy and the understandability of the model's code. If the professionals consider the models to be correct, it implies that Acadela has the potential to model CPs in diverse medical fields. Additionally, if the model code is comprehensible to medical experts, they can discuss the logic and structure of CPs with modelers. As a result, the DSL can serve as a shared artifact to foster communication between medical and technical experts, hence improving the quality of CPs.

Research Question 4

Do **modelers** regard the *DSL* and the *development environment* **user-friendly** and **learnable** when modeling CPs?

Besides the capability of modeling CPs, the DSL shall be *user-friendly* to **modelers**, thus improving their user experience and productivity. Therefore, our study explores the usability of Acadela from the perspective of technical staff working in medical facilities or research institutions. In the evaluation, the participants experience the features of Acadela by modeling a CP and debugging another faulty CP from its error messages. Next, they gave us feedback regarding the DSL learnability, user-friendliness, and applicability to model CPs in e-Health applications.

1.3 User Study

Our evaluation method resembles the user study of Faber (2019), which applied the recommendations of conducting user studies by Robert K. Yin (2009). We first define the research questions (**RQ3** and **RQ4**) regarding expressiveness and usability. Then we identify the target participants for each evaluation. Next, to measure the evaluation outcome, we design assessment tasks and questionnaires to collect quantitative and qualitative feedback from the experts. We then set up pilot tests with computer science research assistants to estimate the cognitive workload and appropriateness of the tasks. The decisive phase is to conduct the evaluation and gather feedback from the participants. Finally, we analyze the data to derive conclusions and implications from the assessments.

Expressiveness Evaluation: This assessment explores the potential of Acadela in modeling generic CPs. To ensure the practical value of the study, we only consider medical professionals working in healthcare facilities. Therefore, we contacted six medical professionals from five medical departments and collected five CPs from their fields of expertise. Next, we modeled the CPs with Acadela and interviewed the professionals to evaluate the accuracy of the constructed CPs. The result shows that medical professionals consider the CPs to be accurate and can be beneficial to their daily routines. However, they suggest that the DSL can reveal further practical value if our study models treatment variations, i.e., alternative care paths triggered in specific conditions. Additionally, most medical professionals feel confident making small changes in the reconstructed CP but often lack the time and technical background to model it.

Usability Evaluation: To study the user-friendliness of Acadela, we conducted field experiments with technical staff working in the healthcare industry or medical research institutions. The participants developed a hypertension CP using Acadela and gave us quantitative and qualitative feedback regarding the user-friendliness, learnability, and applicability of the DSL. The result shows that, in general, the technical staffs consider the syntax of Acadela to be straightforward, learnable, and applicable to model CPs. However, participants with limited experience in programming stated that more training is needed to familiarize themselves with the syntax.

1.4 Dissertation Structure

From the motivation of developing a textual DSL for modeling CPs in various e-Health systems, the remainder of this Dissertation presents its foundational knowledge, implementation, and findings in the following seven chapters:

Chapter 2: Foundations describes the fundamental knowledge regarding the definition of CP, Case Management, DSL development, and the textX DSL design tool applied in our research.

Chapter 3: Related Work presents existing graphical and textual DSLs for modeling CPs. Our study describes the capabilities, strengths, and research gaps that Acadela aims to bridge for each DSL.

Chapter 4: Smart Adaptive Case Management (SACM) introduces SACM as the target e-Health system that executes CPs compiled from the Acadela code. This chapter explains the motivation, features, architecture, and CP meta-model of SACM as the subsystem that models, manages, and executes CPs in the CONNECARE, a European-funded e-Health system specialized in practicing integrated care.

Chapter 5: Language Design explains the design principles applied in creating Acadela, the concrete syntax representation of SACM CP modeling concepts, and the constraint validation of the language.

Chapter 6: Implementation describes the development of the Acadela system, including its IDE, compiler, syntactic and semantic constraint validators, and CP visualization.

Chapter 7: Evaluation presents the results, implications, and limitations derived from our expressiveness and usability assessments with care professionals and technical staff, respectively.

Chapter 8: Conclusion summarizes the findings as answers to the hypothesis and research questions formulated in Section 1.2. Furthermore, our study proposes possible directions for future work to cover the gaps in the current research.

2

Foundations

This chapter presents the fundamental concepts of the motivation, design, and implementation of the Acadela DSL. The first section establishes the definition of CP, including its characteristics, incorporated elements, and goals. Afterward, the second section explains Adaptive Case Management (ACM) as the workflow management mechanism to execute CPs in e-Health systems. Next, the third section summarizes the concepts, nature, and elements of the Case Management Model and Notation (CMMN). Finally, the last two sections explain the definitions and motivation for applying Domain-Specific Language (DSL) and Compiler-compiler language in constructing Acadela.

2.1 Clinical Pathways (CP)

Zander et al. (1987) at the New England Medical Centre first used the term "clinical pathway" in 1985. Since then, the medical field has been extending CP with multiple definitions and alternative terms, such as *critical paths*, *critical pathways*, *care paths*, *care maps* (Every et al., 2000), *integrated care pathways*, and *guidelines* (De Bleser et al., 2006). For example, De Bleser et al. (2006) identified 84 different definitions of CP from their literature survey during the 2000-2003 period. After analyzing various research on CPs (Campbell et al., 1998; Panella et al., 2003; De Bleser et al., 2006), we derive one summary of CP definition as follows:

Clinical Pathway Definition:

Clinical Pathways are multidisciplinary patient-care management plans that define *care goals* with the *process* and *timing* necessary to achieve such goals with *optimal efficiency*. (Campbell et al., 1998; De Bleser et al., 2006; Vanhaecht et al., 2006)

The following subsections recapitulate the goals and characteristics of CPs.

2.1.1 Goals

Care Efficiency Improvement The primary purpose of applying CPs is to increase *care efficiency* (Every et al., 2000; De Bleser et al., 2006; Panella et al., 2003) by reducing the cost of treatment, optimizing resource utilization, and reducing treatment time using optimal care procedures. This goal arises from one critical concern in healthcare is reducing treatment costs without diminishing the care quality (Wentworth & Atkinson, 1996). To solve this issue, CPs with standardized procedures can simplify diagnostic activities by defining the conditions for taking laboratory examinations (Panella et al., 2003, p. 513). For instance, patients only take direct chest radiography when they have not received an X-ray scan in the preceding six months (Panella et al., 2003, p. 514). Consequently, the number of clinical examinations decreases, resulting in a reduction in treatment costs. (Panella et al., 2003, p. 515).

Another contribution of CPs in enhancing care efficiency is their support to care process coordination. Since multidisciplinary care professionals develop standardized treatment activities, CPs help medical practitioners better understand their responsibilities, enhance process's learnability by sharing information, and support the integration of segments in a healthcare system (Panella et al., 2003, pp. 509-510).

Care Quality Enhancement As standardized treatment procedures, one of CP's primary goals is to improve patient outcomes (Every et al., 2000; De Bleser et al., 2006, p. 560) and satisfaction. CPs can achieve this goal by practicing a patient-focused vision as a care priority (Vanhaecht et al., 2009), complying with standardized medical practices, and receiving continuous quality improvement (De Bleser et al., 2006, p. 560; Vanhaecht et al., 2009, p. 786). As a result, CPs can follow the best standard of practice to provide high care quality and minimize outcome variation in treatment processes (Panella et al., 2003).

Evaluation Another purpose of using CPs is to evaluate the variation and outcome of treatment processes (Panella et al., 2003, p. 60; Yan et al., 2017). Although CPs contain predefined, predictable, and standardized care procedures, variances are still unavoidably possible because patients can have unique medical complexities. Furthermore, the subjective initiatives of health-care experts, patients, and their families after applying CP to the care process also contribute to the process variation (Ye et al., 2009). By comparing the standardized care procedures in CPs with the patient's progression from their Electronic Medical Records (EMR), one can identify whether the given care deviates from the expected treatment path. (Yan et al., 2017).

For example, Yan et al. (2017) modeled a CP of unstable angina pectoris (UA) in BPMN and applied it to two datasets containing 888 (only UA diagnosis) and 1608 patients (UA diagnosis and complications). The result shows that Electrocardiogram (ECG) monitoring and examination were missing in both groups of patients. Additionally, they witness an increase in the dose of beta-blockers medicine. These activities may not be performed or recorded in the EMR system (Yan et al., 2017, p. 316), but the variance analysis of the CP model reveals the missing care activities. Therefore, medical experts can locate and redesign their treatment process to efficiently and effectively optimize the care procedure.

Documentation The above variance analysis study demonstrates another essential role of CP in collecting relevant medical data. In general, variance in CPs is the result of omitted, late, or incorrectly performed actions (Every et al., 2000, p. 462). CPs document these series of time-associated actions, resulting in an abundance of data points that can overwhelm the variance analysis procedure (Every et al., 2000). However, this issue is resolvable with computer-assisted pathway analysis (Every et al., 2000, p. 462; Hyett et al., 2007). Thanks to records of positive and negative variances, like early discharge and unplanned surgery return, medical experts can form a collection of evidence to improve the system, clinical practice, and delivery of care service (Hyett et al., 2007).

2.1.2 Characteristics

From an intensive analysis of 82 articles, De Bleser et al. (2006) identified the most outstanding attributes of CPs as follows:

2 Foundations

1. **Homogeneous patient group:** The care of a particular population of patients with a (diagnosis-) specific clinical problem.
2. **Multidisciplinary team:** Diverse medical professionals (e.g., physicians, nurses, administrative staff) are responsible for patient care.
3. **Time scale:** The length of hospital stay for the patient or the optimal timeline of the care procedure.
4. **Inventory of actions:** the procedures, events, or essential components of care.

Other characteristics with moderate mentions are *patient care management*, *care efficiency*, *care standardization*, *sequence in the CP*, and *variance analysis*. *Care standardization* refers partly to uniformity of care practice that reduces variability while attains the minimum level of care.

In another study of CP classification, Kinsman et al. (2010) developed the following five criteria by conducting a literature review and testing them against 260 papers:

1. The intervention was a *structured multidisciplinary* plan of care.
2. The intervention was used to channel the translation of *guidelines* or *evidence* into *local structures*.
3. The intervention detailed the *steps* in a course of treatment or care in a plan, pathway, algorithm, guideline, protocol or other ‘inventory of actions’.
4. The intervention had *time-frames* or *criteria-based progression* (that is, steps were taken if designated criteria were met).
5. The intervention aimed to *standardize care* for a specific clinical problem, procedure or episode of healthcare in a *specific population*.

The author derived the above criteria from three articles that comprehensively review the characteristics of CPs. Table 2.1 summarizes the identified CP properties from the three papers.

De Bleser et al. (2006)	Campbell et al. (1998)	Vanhaecht et al. (2006)
Guides care management for a well defined group of patients for a well defined period of time	Structured multidisciplinary care plan	Facilitate variance management
States goals and key elements of care based on evidence and best practices	Detail essential steps in care of patients with a specific clinical problem	Support multidisciplinary care
Sequences the actions of a multidisciplinary team	Facilitate translation of national guidelines into local protocols	Support evidence-based clinical practice
Allow documenting, monitoring and evaluating of variances	Help communication with patients by providing a clearly written summary of care	

Table 2.1: Characteristics of CPs derived from the three articles

Summary From the above research work, the fundamental features of CPs are 1) *Evidence-based clinical practice* to treat *homogeneous patient population* with a particular health problem, 2) *Multidisciplinary care teams* responsible for 3) specific *sequence of care activities*, which

can be 4) *time-constrained* to achieve optimal timing for the treatment. Finally, 5) *medical data documentation and monitoring* are necessary for evaluating the variance of care practices.

2.2 Adaptive Case Management (ACM)

Software engineering and business process management studies classify workflow management systems into two categories. On the one hand, Van der Aalst et al. (2005, p. 8) classify business processes into *Workflow Management* and *Case Handling*. On the other hand, Swenson (2013) and Motahari-Nezhad and Swenson (2013, p. 265) distinguish two approaches for case management: **Production Case Management (PCM)** and **Adaptive Case Management (ACM)**.

2.2.1 Concerns of Production Case Management

The similarity between PCM and Workflow Management is that knowledge workers execute a set of actions specific to the case scenario (Van der Aalst et al., 2005, p. 4; Swenson et al., 2012, p. 110). Technical experts can define these sequences of actions in advance (Swenson et al., 2012, p. 110) at the design time (Van der Aalst et al., 2005, p. 2). However, when handling a case in PCM, knowledge workers cannot significantly modify their activities as they are restricted to predefined tasks and options (Motahari-Nezhad & Swenson, 2013, p. 265; Van der Aalst et al., 2005, p. 2).

Application Context The design of PCM can handle scenarios with a certain level of unpredictability while allowing sufficient flexibility in selecting the necessary actions (Swenson et al., 2012, p. 109; Van der Aalst et al., 2005, p. 2). These scenarios contain numerous variations among individual cases, so it is impossible to define a fixed procedure. Instead, knowledge workers choose a dedicated group of actions to process the case (Swenson et al., 2012, p. 110).

Adaptability Concerns Developing PCMs raises several concerns in constructing an adaptive process. First, designing a simplistic model cannot handle the vast number of variations arising at runtime; nevertheless, if the model attempts to cover all possible exceptions, it significantly increases the complexity of maintaining and managing the model (Van der Aalst et al., 2005, p. 2). The second drawback is that developers follow a standard application development lifecycle, i.e., program, test, then deploy, which limits the ability of case managers to modify the workflow process (Swenson et al., 2012, p. 113).

Furthermore, case managers and knowledge workers need to decide their actions in unprecedented scenarios, the PCM design limits their available options in these circumstances. In the context of healthcare, this situation can be fatal if doctors cannot execute radical treatment (Swenson et al., 2012, p. 113) or request a different medical test (Van der Aalst et al., 2005, p. 4) because the defined model does not offer these options. Therefore, handling cases in medical fields considers applying ACM because it enables caseworkers more flexibility to adapt their treatment process without potentially involving technical experts.

Adaptive Case Management Definition:

"Systems that are able to support decision making and data capture while providing the freedom for knowledge workers to apply their own understanding and subject matter expertise to respond to unique or changing circumstances within the business environment" (Swenson & Palmer, 2010).

2.2.2 Suitability for Clinical Pathways Modeling

Flexible Case Execution and Modification In ACM, there exists no predefined control flow. Instead, the sequence of activities materializes during the case handling based on the care goals (Herrmann & Kurz, 2011, p. 83). ACM possesses this ability because each case consists of a case workspace and objectives. In addition, each workspace contains a process comprising tasks necessary to achieve the case objectives. Figure 2.1 illustrates the key concepts of adaptive case construction in ACM (Kurz, 2013).

By leveraging a simple hierarchy of tasks instead of complex process networks like in BPMN 2.0, ACM helps knowledge workers to adapt the process at runtime (Swenson et al., 2012, p. 109; Kurz, 2013). This feature empowers knowledge workers to adapt and improve the case process as the situation arises (Swenson et al., 2012, p. 109; Kurz, 2013). As a result, knowledge workers do not have to follow a static set of predefined processes, which may be unsuitable for exceptional or unusual scenarios.

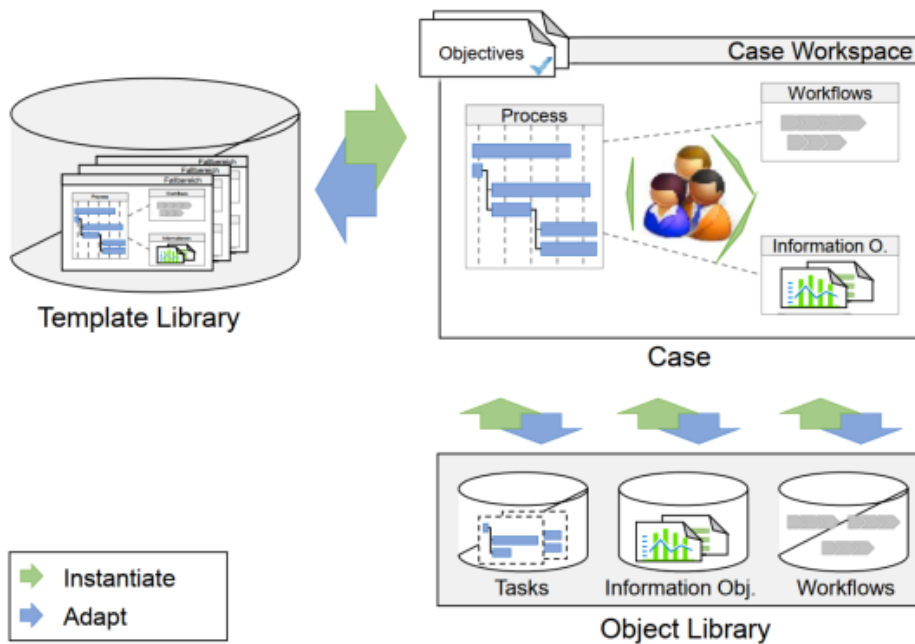


Figure 2.1: ACM Key Concepts.

The flexible definition, modification, and management of case templates and instances in ACM support CP modeling significantly. The reason is that HISs shall incorporate relevant stakeholders and variations during their execution in work practice for a better care quality (Hitt & Tambe, 2016, p. 838; Avgar et al., 2018). Empowering knowledge workers to execute exceptional tasks

for unprecedented scenarios is recommended in CP execution (Campbell et al., 1998, p. 135), and especially vital in handling life-or-death situations. This ability enables medical professionals to deviate from the standard pathway (Campbell et al., 1998, p. 135), or conduct radical treatments that are hopefully effective but not predefined in the system to save a patient's life (Swenson et al., 2012, p. 113). Flexible modification of CP at runtime also supports a practice, which encourages adapting the new variance of the CP after a careful multidisciplinary auditing (Campbell et al., 1998, p. 136).

Caseworker Competency Advancement ACM facilitates a faster response to organizational or routine changes and higher proficiency in processing unpredictable situations because knowledge workers without modeling or programming skills can modify the process themselves at runtime (Hauder et al., 2014a, p. 2). The result is a continuous promotion of innovative service delivery (Swenson et al., 2012, p. 113) and learning effect in the organization (Panella et al., 2003, pp. 509-510; Swenson & Palmer, 2010).

Modeling CPs using ACM can further enhance the experience and expertise of medical staff because the execution paths and control flows in CPs show how previous or particular cases were processed. Consequently, these data from the past serve as the recommended method to handle the cases for knowledge workers (Hauder et al., 2014a, p. 4).

Authorized Case Execution Clinical processes can involve a multidisciplinary team of medical professionals (e.g., physicians, nurses, or administrative staff). During the treatment process, each expert can only conduct their designated tasks without accessing the data or activities that are not assigned or eligible for their role. ACM can define such an access control mechanism thanks to the following perspectives (Kurz, 2013, p. 88):

1. **Common Organization Perspective (COP)**: refers to case objects, case goals, and basic case structure shared by all participants. Thanks to COP, ACM can define activities or data accessible to *all knowledge workers in a case*.
2. **Private Organizational Perspective (POP)**: contains the best practices that an organization or its members conduct to reach the case objectives. Defining treatment tasks with restricted access to a *particular group* of medical professionals is possible with POP.
3. **Private Agent Perspective (PAP)**: supports access control such that certain activities or case information are only executable to *individual caseworkers*.

One goal of CPs is to provide a common treatment plan view for medical experts to understand their various roles in the care process (Pearson et al., 1995; Every et al., 2000, p. 462). As a result, the involvement of medical experts in the care process is crucial for developing and implementing CPs (Every et al., 2000, p. 462). For these reasons, ACM with the fine-grained access control mechanism enables multidisciplinary teams to define restrictions at an organization, group, or individual level for each treatment activity and case datum. This coordination in access control definition is also effective in preventing opinion-based variations created by having only one professional team leading the CP development (Wooster & Forthman, 1996; Anders et al., 1997, pp. 14-15; Bailey et al., 1998, p. 37).

2.2.3 Challenges

Technical Concerns: Encouraging knowledge workers without programming experience to adapt and instantiate case templates at runtime requires a flexible execution environment. Specifically, this environment should simultaneously enable modelers to create and caseworkers to interact with case templates (Di Ciccio et al., 2015; Marin et al., 2016, p. 2). As a result, the design requirements for this environment shall seamlessly address the following essential concerns of ACM development (Hauder et al., 2014a, pp. 8-10):

1. **Data integration:** Data are the essential ACM element that provides context for determining the process execution. Concurrent modification of case data from multiple caseworkers can cause inconsistent states. ACM systems can prevent this by appropriately handling a shared memory, locking of data objects, and linking of data objects and processes.
2. **Knowledge work empowerment:** Providing autonomy for knowledge workers is an outstanding characteristic of ACM. Therefore, ACM systems concern guidance techniques to recommend the next steps for knowledge workers to handle new cases effectively (Hauder et al., 2014a, p. 4). Additionally, advanced collaboration is crucial in scenarios requiring the inclusion of individual expertise, experience, and collective judgment to process the case (Hauder et al., 2014a, p. 4). Furthermore, expert knowledge in case execution is significant for leveraging the emerging design of case templates.
3. **Authorization and Role Management:** With data integration into the process, some attributes in the case form are not accessible to users, depending on their role and the case state. Additionally, ACM systems should enforce constraints and rules to forbid case template modification to specific roles.
4. **Workflow Management Foundation:** Besides adaptable case management and rule enforcement, ACM systems need to support routine paths and process patterns by integrating with traditional workflow management or other methods. However, the adaptation of process instances at runtime might trigger inconsistent states or deadlocks in the system. This problem can be solvable by verifying process models in case management.
5. **Knowledge Storage and Extraction:** Data in SACM, such as contextual information from conversations or implicit knowledge from execution traces of process instances, can be primarily unstructured. Since ACM should support knowledge-intensive processes and not overload case workers with numerous insignificant information, its system shall possess an efficient and user-friendly mechanism to store and extract relevant knowledge.

2.3 Case Management Model and Notation (CMMN)

CMMN is the definition of a common meta-model and notation for case modeling and visualization (Object Management Group, 2022). The developer of CMMN is Object Management Group (OMG), an international, open membership, and for-non-profit consortium that develops technology standards for industries (Object Management Group, 2022). OMG published the first version of the CMMN specification in May 2014. At the time of this research, the latest version of CMMN is 1.1, published in 2016. (Object Management Group, 2016)

The purpose of CMMN is to support the modeling of unpredictable and evolving processes, which expects an indeterministic sequence of activities during execution (Object Management Group, 2022). Another goal of CMMN is to establish an interchangeable format to exchange case

models across various tools. The following subsections further explain 1) the CMMN applicability to support ACM and CP modeling and 2) the concepts and notation of the framework.

2.3.1 Applicability to ACM and CP Modeling

The vision of CMMN seamlessly aligns with ACM since their common objective is to support the management of cases that cannot predefine activities in advance. In healthcare services, the processes of medical treatments and diagnoses are exemplars of such scenarios. A remedy applied to a patient cannot thoroughly predetermine its workflow prior to the treatment (Michel, 2020, p. 15). The reason is that healthcare professionals execute medical activities (e.g., diagnoses, medical tasks, or laboratory tests) depending on the patients' medical conditions (Kinsman et al., 2010, p. 2) or problems (Campbell et al., 1998, pp. 133-134). However, each patient's treatment needs to conduct the same tasks (Michel, 2020, p. 15). For example, a CP of cardiac catheterization requires medical professionals to complete a checklist of actions, investigations, and patient's condition assessment before discharge (Campbell et al., 1998, p. 134). CMMN can model both dynamic and structured sequences of activities thanks to its definition of DiscretionaryItems and PlanItems as Figure 2.2 illustrates (Object Management Group, 2016, p. 7).

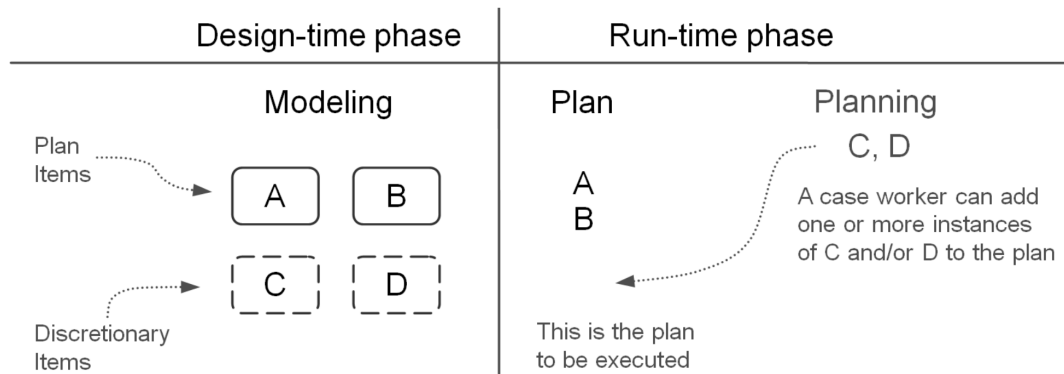


Figure 2.2: Design-time phase modeling and runtime phase planning

A typical *case* has *design-time* and *runtime* phases. During the *design time*, knowledge workers (medical experts) and modelers collaborate to define *PlanItems*, which refer to activities required to execute a predefined process segment. For instance, every treatment ends with filling out a discharge form to assess the patient's condition. Additionally, the case includes *DiscretionaryItems*, which represent activities besides the compulsory *PlanItems* that knowledge workers can apply at their discretion to achieve the case goal. Thanks to *DiscretionaryItems*, medical experts can adaptively execute extraordinary tasks in unprecedented situations, such as requesting a specific laboratory test that has not been conducted in any previous treatment.

Another positive influence of *DiscretionaryItems* is their support for the evolvement and improvement of CPs as medical experts can decide and share the effective combination of activities to handle a specific disease variation. A typical example of treating a particular variant improves the experience and expertise of knowledge workers, as they are familiar with the recommended steps should a similar scenario happen in the future. Furthermore, over time, the team can include non-mandatory but frequently-used activities as compulsory *PlanItems* of the Case Model.

To further demonstrate the applicability of CMMN in modeling CPs following the ACM approach, the next subsection describes CMMN notations and how they can be applied to construct CP elements.

2.3.2 Concepts and Graphical Notations

Task A CMMN Task represents activities executed during the handling of a Case. Every Task has an *isBlocking* attribute. When set to true, the Case will wait for the Task to finish. Otherwise, the Case completes the Task upon instantiation without waiting for its accomplishment (Object Management Group, 2016, p. 45).

Additionally, every Task has its Discretionary version (e.g., Discretionary HumanTask) to express that a particular activity is not compulsory to complete a process (Object Management Group, 2016, pp. 64-67). However, knowledge workers can execute the Task when handling the Case at their discretion. CMMN categorizes Tasks into four types:

1. HumanTask: expresses activities conducted by knowledge workers (Object Management Group, 2016, p. 46-47). Manual tasks performed by healthcare professionals, such as discharge, prescription, or taking laboratory tests, belong to this category.
2. ProcessTask: triggers the execution of a business Process (Object Management Group, 2016, p. 47-48). ProcessTasks are necessary when ACM systems need to call a function or service from an internal HIS of a medical institution or an external system. For instance, to retrieve the complications or interactions between two medications. In this example, the task inputs are the drug names, and the output is the side effects or contraindications created by the medications.
3. CaseTask: creates an instance of another Case (Object Management Group, 2016, pp. 49-50).
4. DecisionTask: invokes a defined CMMN Decision of the Case. A Decision takes inputs from a DecisionTask and returns its outputs to a DecisionTask (Object Management Group, 2016, p. 50). CMMN supports defining an expression to select a Decision (Object Management Group, 2016, p. 51); thus, one can declare conditional statements to decide which Decision is valid. This Task type can be helpful when knowledge workers need to retrieve care recommendations based on the patient medical conditions.

Figure 2.3 shows the CMMN graphical notations of the four Task types.

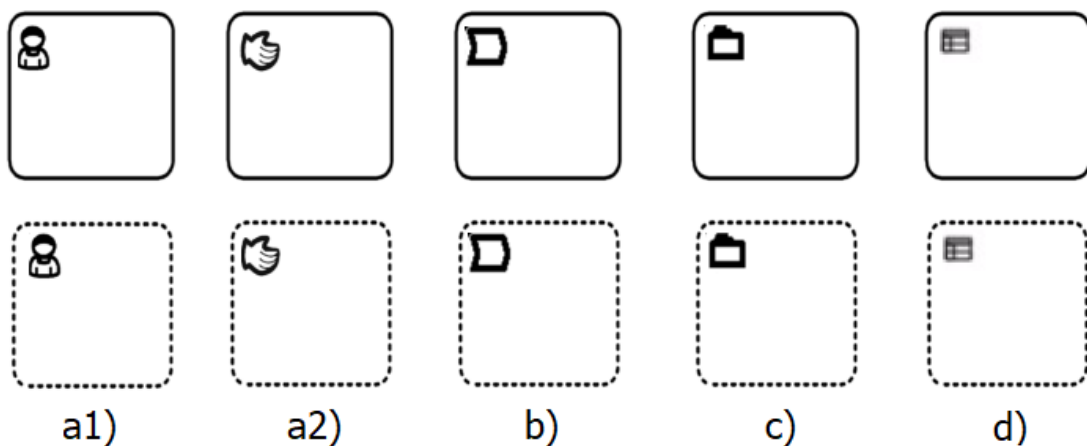


Figure 2.3: Graphical Notations of four Task types expressed in each column with the bottom row shows their Discretionary version. a1) Non-blocking HumanTask, a2) Blocking HumanTask, b) ProcessTask, c) CaseTask, d) DecisionTask.

CaseFileItem CMMN represents Case Information as a CaseFile (Object Management Group, 2016, p. 21). Each CaseFile contains CaseFileItems, which denote data using any type of data structure (Object Management Group, 2016, p. 22). Therefore, a CaseFileItem can contain a single or a hierarchy of data objects. As a result, CaseFileItems can store folders, a hierarchy of folders, or (XML) documents (Object Management Group, 2016, p. 22). Furthermore, it is possible to specify the metadata of each CaseFileItem and its parent or child CaseFileItems (Object Management Group, 2016, p. 23). CMMN associates only one CaseFile to a Case.

Case Management in a clinical context can leverage CaseFileItems to store a collection of medical documents such as a patient’s profile, guidelines, or necessary forms to accomplish the treatment. Figure 2.4 illustrates a graphical notation of CaseFileItem in CMMN (Object Management Group, 2016, p. 65).



Figure 2.4: Graphical Notations of a CMMN CaseFileItem.

EventListeners To model time constraints and manual events, CMMN offers TimerEventListener and UserEventListener to enable, activate, or terminate Stages or Tasks (Object Management Group, 2016, p. 25).

TimerEventListener represents the elapse of time (Object Management Group, 2016, p. 26). In clinical processes, TimerEventListener enforces a time constraint that an *activity* or *phase* should finish or repeat within a specific amount of time.

UserEventListener captures events created by knowledge workers (Object Management Group, 2016, p. 28). In order to influence the workflow of a Case, a user event can directly change the state of Stages or Tasks (Object Management Group, 2016, p. 28). Another possibility is to modify the CaseFileItems, which stores all the data collected by the Case (Object Management Group, 2016, p. 28). Stages or Tasks can use CaseFileItems data of any type as inputs or outputs of its execution. (Object Management Group, 2016, pp. 21-22).

Figure 2.5 illustrates the graphical notations of TimerEventListener and UserEventListener in CMMN. EventListeners has a circle shape and a clock or human icon to respectively depict TimerEventListener or UserEventListener (Object Management Group, 2016, p. 69).

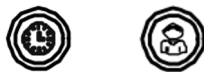


Figure 2.5: Graphical Notations of TimerEventListener (left) and UserEventListener (right).

Sentry CMMN introduces the Sentry element to trigger events and conditions among CMMN Case Elements. This ability enables Sentry to establish a dependency among Tasks or Stages. Sentry expresses a condition with an *"if<condition>"* syntax. An event declaration follows an *"on <event>"* structure, with *<event>* referring to one state in the lifecycle of a CaseItem, e.g., *"on activate"*. A combination of an event and condition has a *"on <event> if <condition>"* syntax.

2 Foundations

These abilities enable Sentry to define 1) Entry Criteria to enforce the condition for triggering a Task or Stage from another; 2) Exit Criteria to dictate the condition for finishing a CaseItem.

Sentry is essential in handling variations and realizing adaptations to Case handling. For example, in clinical processes, knowledge workers may need to adapt (e.g., activate, complete, or skip) an alternative activity or phase based on specific medical conditions or external events.

Figure 2.6 demonstrates how a Sentry establishes a dependency connection between two Tasks or two Stages (Object Management Group, 2016, pp. 70-71). In addition, Figure 2.7 demonstrates how CMMN visualizes a disjunction (OR) and conjunction (AND) relation among Tasks (Object Management Group, 2016, p. 71).



Figure 2.6: CMMN Sentry establishes a dependency between two Tasks (left) or two Stages (right). The white diamond represents an Entry Criterion, while the black diamond denotes an Exit Criterion.

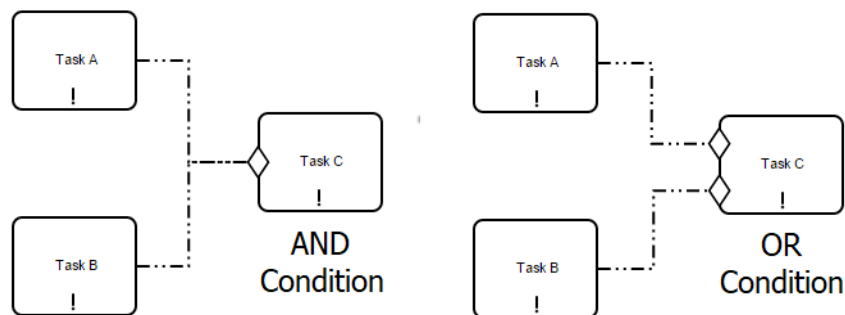


Figure 2.7: CMMN depicts an AND (left) or OR (right) relationship among Tasks using Sentry as Entry Criteria.

Milestone To mark a checkpoint for progress evaluation, CMMN defines a Milestone element to represent an achievable goal in the process. A Milestone includes no Task, but the accomplishment of Tasks or deliverables (e.g., the value of a CaseFileItem) typically leads to the Milestone achievement (Object Management Group, 2016, p. 28). The condition(s) to reach a Milestone is expressible by defining (multiple) Entry Criterion applied to a Milestone.

Medical teams can apply Milestone to mark or set care objectives at different phases of the treatment. Figure 2.8 illustrates the notation of a Milestone and a Milestone with one Entry Criterion in CMMN (Object Management Group, 2016, p. 68).



Figure 2.8: A Milestone (left) with one Entry Criterion (right) notations in CMMN.

Stage CMMN Stage is equivalent to a phase of a process. In this regard, a Stage can group Tasks (activities) and Stages (treatment phases) that belong to the same medical context (e.g.,

2 Foundations

Patient Discharge, Diagnosis, or Prescription). Furthermore, a Stage can hold EventListeners to model timing constraints or triggers of the included Task or Stage.

Additionally, a Stage can contain both plan and Discretionary Tasks or Stages (Object Management Group, 2016, p. 37). The DiscretionaryItems have entry and exit criteria to control their activation and termination (Object Management Group, 2016, p. 41) with conditional expressions as Sentry (Object Management Group, 2016, pp. 32-33).

Figure 2.9 depicts the graphical notation of the CMMN Stage and its Discretionary version (Object Management Group, 2016, p. 61). Clicking the plus button at the bottom expands the Stage and reveals its inner elements. Afterward, the plus icon turns into a minus symbol to signal that the Stage is collapsible.



Figure 2.9: Graphical Notations of a CMMN Stage (left) and Discretionary Stage(right).

CasePlanModel Every Case is associated with only one CasePlanModel that serves as a container for all case elements (Object Management Group, 2016, p. 20), e.g., Stages and Tasks in a treatment. The CasePlanModel can evolve at runtime by incorporating extra Case elements when needed (Object Management Group, 2016, p. 20). Therefore, a multidisciplinary care team can modify existing or insert extra CP elements to adapt their medical interventions. Furthermore, the lifecycle of CasePlanModel is trackable at runtime, enabling medical experts to constantly monitor patient Cases, e.g., whether the treatment is *active*, *pending*, *closed*, *completed*, or *failed*. (Object Management Group, 2016, p. 110).

Figure 2.10 illustrates the graphical notation of a CasePlanModel having a folder shape, with the case name written on the top left rectangle. The folder body contains all the Case elements, such as Stages and/or Tasks (Object Management Group, 2016, p. 59).

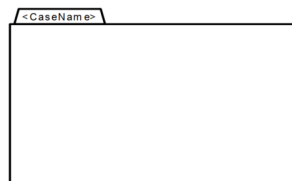


Figure 2.10: CMMN Graphical Notation of a CasePlanModel. The folder body includes all Case elements

Decorators To express the behaviors of PlanItems and DiscretionaryItems, CMMN applies Decorators, which are symbols to denote the supported behavior pattern of a particular CMMN element. Decorators' graphical notation resides on the bottom center of a CMMN element. Figure 2.11 demonstrates the list of applicable Decorators for each CaseItem type (Object Management Group, 2016, p. 79). Note that the CaseItem represents both its Plan and Discretionary version. For example, A Stage Item refers to both the Stage and Discretionary Stage; Decorators applicable to a Task means that the Plan and Discretionary HumanTask, ProcessTask, CaseTask, or DecisionTask can also use these Decorators, except when specified otherwise. CMMN provides the following Decorators:

2 Foundations

1. **PlanningTable:** defines the context for planning a Stage or HumanTask. A PlanningTable of a Stage includes the executable (Discretionary) Tasks and nested (Discretionary) Stages (Object Management Group, 2016, p. 38). While a PlanningTable of a HumanTask contains 1) the follow-up Stages and Tasks of the HumanTask (Object Management Group, 2016, p. 46), or 2) Discretionary HumanTask or Tasks that are relevant to the execution of the HumanTask (Object Management Group, 2016, p. 47).
2. **Entry and Exit Criteria:** See the Description of 2.3.2.
3. **AutoComplete:** Having no AutoComplete Decorator means a Stage or Case requires case-workers to complete it manually (Object Management Group, 2016, p. 38).
4. **ManualActivation:** Defines the conditions for manually or automatically starting a Task or Stage once it is enabled (Object Management Group, 2016, pp. 51-52).
5. **Required:** Specifies that a Task, Stage, or Milestone has a condition for completing or terminating itself before the parent Stage can finish. In other words, a Stage can only finish when its Required child Task, Stage, or Milestone completed or aborted (Object Management Group, 2016, pp. 51, 53).
6. **Repetition:** States the condition for a Task, Stage, or Milestone to be repeated. (Object Management Group, 2016, pp. 51, 53).

Decorator Applicability	Planning Table	Entry Criterion	Exit Criterion	AutoComplete	Manual Activation	Required	Repetition
CasePlanModel 	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
Stage 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Task 	HumanTask only	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
MileStone 		<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
EventListener 							
CaseFileItem 							
PlanFragment 							

Figure 2.11: CMMN Applicable Decorator to Caseltems (Object Management Group, 2016, p. 79).

2.4 Domain Specific Language (DSL)

The focus of Enterprise Modeling is constructing and applying conceptual models to describe, analyze, and (re-)design information systems, action systems (e.g., business process models or strategy models (Frank, 2014, p. 3)), along with other organizational aspects (Frank, 2014, pp. 2-3; Sandkuhl et al., 2014; Heß et al., 2015). These information and action systems involve multidisciplinary teams of experts from different backgrounds and professions (Frank, 2014, p. 2). Therefore, abstractions (models) representing a particular view of the systems are necessary to provide an insight into the process. Moreover, the model should apply domain concepts corresponding to the experts' background as a common reference to foster communication among multidisciplinary teams (Frank, 2014, p. 3). Creating conceptual models requires expressing domain concepts through a modeling language (Siau & Rossi, 2011, p. 251).

2.4.1 Modeling Language Elements

The foundations of modeling languages are their syntax and semantics. Therefore, designing a modeling language concerns the definition of three elements below:

1. **Abstract syntax** dictates the rules to define language concepts for constructing a model syntactically (Frank, 2011a, pp. 26-27). It is similar to syntax rules of grammar in human language, e.g., subject-verb agreement; past, present, or future tenses.
2. **Concrete syntax** defines the symbols to express language concepts based on the abstract syntax (Frank, 2011a, p. 27). An example is in Java, to express an *access modifier* (an abstract syntax rule) of a class's function or variable, one can use `public`, `private`, `protected`, or no character as textual symbols of Java concrete syntax (Oracle, 2022).
3. **Semantics** defines the meaning for (formally) interpreting modeling concepts (Frank, 2011a, p.27; Harel & Rumpe, 2004, p. 67). The language semantic supports the analysis, refinement, manipulation, and evolution of the models (RWTH Aachen Software Engineering Chair, 2022). For example, in several programming languages (e.g., C, Java, Javascript), the syntax `if (a < 2) func1();` represents a conditional expression to instruct that if variable `a` is less than 2, then execute the function `func1()`.

Constructing a conceptual model can follow two approaches, 1) using a General Purpose Modeling Language (GPML) or 2) Domain-Specific Modeling Language (DSML). The next sections describe the concerns of using GPMLs and present the benefits of leveraging DSMLs over GPMLs.

2.4.2 Advantages of Domain Specific Modeling Languages

DSMLs intend to model concepts exclusively belonging to a particular domain by leveraging its technical terms (Heß et al., 2015, p. 3). Thanks to this characteristic, developing a modeling language as a DSML offers several advantages compared to using GPML:

1. **Enhance productivity:** in GPML, the construction of every model element has to be done from scratch and relies on primitive constructs such as classes, attributes, or data types (Frank, 2011a, pp. 27-28; Frank, 2013, pp. 5-6). This constraint limits the expressiveness of communication because multidisciplinary teams with diverse background knowledge can only apply a few primitive concepts to explain their model's syntax and semantics.

Meanwhile, DSML removes this barrier as developers can model domain concepts without using primitive constructs (Frank, 2011a, p. 28). Specifically, using meta-languages like *Xtext*¹ or *textX*², developers can define the syntax and grammar rules to declare a domain object with its attributes and behaviors. The meta-language then identifies the domain object type, attributes, and behaviors by parsing the elements based on the grammar rules. Afterward, the meta-language automatically constructs a General Programming Language (GPL) object with primitive concepts. This mechanism can increase productivity as developers can declare or modify domain concepts and model constraints without using GPML constructs, as shown in Figure 2.12. (Frank, 2011a, pp. 28-29)

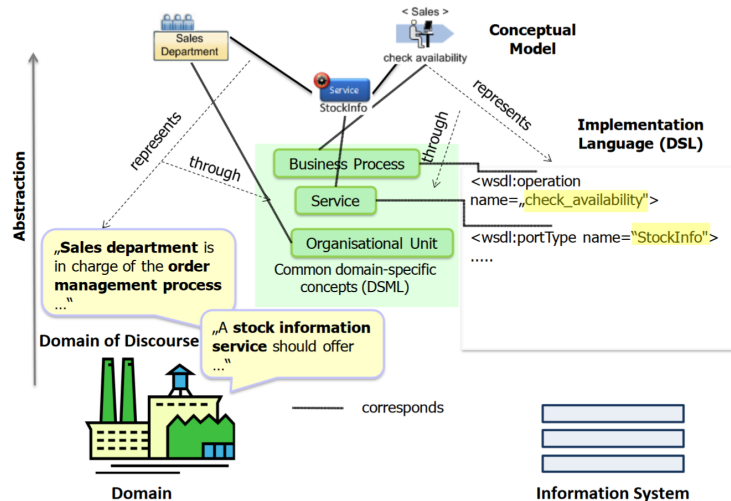


Figure 2.12: Illustration of a DSML mechanism that identifies domain-specific concepts from the business domain and represents the concepts through an implementation language (DSL). (Frank, 2011a, p. 29).

2. **Foster Model Integrity:** Building a modeling language in GPML may pose a risk to integrity because the modeling phase involves the domain experts defining model constraints (Frank, 2013, p. 6). However, the domain experts or model users may be unfamiliar with GPML or GPL concepts as they do not originate from a technical background. Consequently, developers define the constraints using GPL features that the domain experts may not thoroughly understand; hence a flaw can occur due to miscommunication.

In contrast, DSML has an advantage over GPML in excluding the model's inconsistencies (Frank, 2011a, p. 28). In DSML, developers can leverage implicit constraints from domain-specific concepts and DSML constructs to prevent declarations of illegitimate models (Frank, 2011b; Córdoba-Sánchez & De Lara, 2016, e.g., pp. 2-3, 35). Furthermore, DSML developers and domain experts can collaboratively verify these constraints in the code and model as the domain concepts are the common reference to express the restrictions (Frank, 2011a, p. 28).

3. **Foster Communication:** From the previous two benefits, applying DSML implies the possibility of enhancing understanding and discussion of the model concepts and constraints for both developers and domain experts. This advantage is achievable because the expression of various model concepts leverages domain terminology that the model users can fluently command. (Heß et al., 2015, p. 3)

¹**Xtext:** <https://www.eclipse.org/Xtext/>

²**textX:** <https://textx.github.io/textX/3.0/>

The representation of DSMLs can be graphical or textual to deliver the benefits. However, developing DSMLs needs to address several challenges. The following subsections describe the advantages and concerns in designing graphical or textual DSMLs.

2.4.3 Graphical Domain Specific Languages

Concept Graphical DSMLs leverage visual elements to express the elements and relationships of domain concepts (Frank, 2013, p. 7; Shen et al., 2021, p. 3121). Therefore, users typically interact with an editor to create or modify their model (Frank, 2013, p. 6). These interactions include, but are not limited to, drag-and-drop elements, clicking on a visual element to create and edit a model object, and typing to define, modify, or search for model elements.

Advantages Modeling elements using graphical symbols offers several benefits:

1. **Enhance Usability:** Displaying concepts in graphical notations can increase the understandability and readability of the models to their users (Frank, 2010, p. 1). Logical and hierarchical illustrations of model elements (Wienands & Golm, 2009, p. 458) contribute to the user-friendliness and learnability of the DSL (e.g., Hermans et al., 2009, p. 433). Specifically, graphical DSLs reduce the learning effort as users do not have to thoroughly understand the declaration and semantic rules (e.g., Schlee & Vanderdonckt, 2004, p. 2).
2. **Foster Communication:** Visualizing models provides a direct and overarching insight into the solutions. As a result, this diagrammatic representation facilitates communication and collaboration in a more straightforward and illustrative way than textual DSLs.
3. **Error prevention:** Graphical DSLs handle syntactic and semantic constraints on both the graphical and conceptual levels (Frank, 2013, p. 6). Therefore, preventing faults is possible by prohibiting users from declaring erroneous elements or process flow graphically (Hermans et al., 2009, p. 433; Frank, 2011a, pp. 35-36).

Concerns Aside from the above benefits, using graphical DSLs to model CPs faces the following challenges:

1. **Complex Extension:** Using a standard process model framework like BPMN lacks the visual elements to represent the domain concepts (Braun et al., 2014, pp. 1, 5; Heß et al., 2015, pp. 7-8). Therefore, DSL designers must extend the standard framework by modeling the rules and graphical notations as a meta-model (Braun et al., 2014, pp. 5-6). Constructing this meta-model requires DSL developers to learn and use the extension tool of the standard modeling language (Braun et al., 2014, p. 3; Heß et al., 2015, p. 8). In textual DSLs, defining a new language or domain concept is more straightforward. DSL designers directly create a set of rules or sub-DSLs in the grammar using the constructs provided by the DSL definition framework.
2. **Design Elegant UI:** Certain domains possess a complex knowledge base of terminologies, restrictions, and dependency among concepts (Morgan et al., 2018). In these cases, DSL designers encounter the challenges of depicting visual elements for modeling large-scale problems (Shen et al., 2021, pp. 3124-3125). Specifically, Shen et al. (2021, p. 3125) noticed that inappropriate layout arrangement may result in a disordered UI and hidden element details.

2.4.4 Textual Domain Specific Languages

In this form of DSL, *textual characters and symbols* are the core elements of the *abstract* and *concrete syntax*. Similar to graphical DSL, textual DSL needs a *meta-model* containing abstract entities to represent the domain concepts (Jouault et al., 2006, p. 2). Alternatively, textual DSL can define the *abstract syntax* as a *grammar* containing the *syntactic constructs* to express the model elements textually (Harel & Rumpe, 2004, p. 69; Cook et al., 2007, pp. 15-17). The *concrete syntax* is a *textual representation* of the domain concepts based on the *grammar rules* (Cook et al., 2007, pp. 15-16) or *meta-model* (Jouault et al., 2006, p. 2). If the DSL leverages grammar, it can process the concrete syntax using a parser-generator such as Yacc or ANTLR to read the grammar and construct model elements accordingly (Cook et al., 2007, pp. 15-16).

Regarding the semantics of the model, grammar-based textual DSL expresses the semantic domains (e.g., numbers, text, conditions) using a combination of characters, arithmetic operators, or other symbols (Harel & Rumpe, 2004, pp. 66-67). Meanwhile, a meta-model-based textual DSL can map the meta-model definition to another DSL or GPL with execution semantics (Jouault et al., 2006, p. 2). For example, a DSL can map the firing rules of a Petri net to a Java code model (Jouault et al., 2006, p. 2).

Advantages Defining conceptual models using textual elements offers the ensuing benefits:

1. **Fast Editing:** the IDE of a textual DSL can increase productivity and ease of use with facilities such as code completion, text highlighting, and syntax checking (Cook et al., 2007, pp. 16-17; Merkle, 2010). These features prevent fault as the users 1) reuse syntax-compliant templates to define model elements and 2) are aware of unusual errors, warnings, or colors that appear in their code.
2. **Foster Direct Communication:** The textual forms of code with domain-specific terms (e.g., mathematical notations) enables model users to express abstract concepts, and their properties straightforwardly (Shen et al., 2021, p. 3121). Furthermore, the textual code is conveniently distributed to the relevant domain experts as documents; hence they can communicate efficiently. For example, domain experts can refer to a problematic entity, attribute, condition, or expression in the model from their line of code on a (printed) document.
3. **Performance:** Textual DSLs can handle large-scale and sophisticated problems with their concise and domain-specific expression of concepts (e.g. K. J. Brown et al., 2011; Kindlmann et al., 2016; Córdoba-Sánchez & De Lara, 2016, p. 33; Shen et al., 2021, p. 3121).
4. **Ease of Extension:** a textual DSL or a system can consist of sub-DSLs to express a particular concern of the domain concepts. Therefore, according to Riegel et al. (2018, p. 393), each sub-DSL can evolve independently to extend the language. Furthermore, reusing the modeling language's element is possible among different sub-DSLs, e.g., textX allows one meta-model (sub-DSL) to import another meta-model component (Dejanović, n.d.a).
5. **Support Integration:** Since the written codes are textual documents, a version control tool can manage, merge, or find differences in the codes produced by textual DSLs. (Merkle, 2010; Shen et al., 2021, p. 3121). Furthermore, the production of textual outputs enables textual DSLs to work as an internal component of a system. Reusing an existing language's facilities enables efficient construction and flexible customization of the DSL (Günther, 2009, p. 6; Shen et al., 2021, p. 3122).

Concerns : Textual DSLs development experiences the following challenges:

1. **Intuitive Model Overview:** Pure textual DSLs express details of all model elements without visualizing their order and dependency. This deficiency is a disadvantage compared to graphical DSLs that can illustrate workflow or data flow in a highly abstracted, concise, and intuitive manner. (Shen et al., 2021, p. 3122).
2. **Fragmented Systems:** In systems comprising multiple textual sub-DSLs representing different domain concepts, customizing a particular DSL requires changing relevant modules in the system framework and DSL build tools. For example, Rieger et al. (2018, p. 393) examined the modularization of textual DSLs. They discovered that an Xtext DSL depends on five Eclipse projects for unit testing, grammar definition, and integration to the Eclipse editor. If a system uses six DSLs, it needs to manage 30 projects relevant to DSL configuration. Furthermore, developers should balance language features and extensions such that they do not change the core language or limit the reusability of the host language (Rieger et al., 2018, p. 394).
3. **Demand Learning Effort:** Users of textual DSLs need to understand the syntax rules for declaring model elements to represent a domain concept. In contrast, graphical DSLs with an intuitive interface can relieve users from understanding the syntax declaration and semantic rules (Shen et al., 2021, p. 3122). As a result, the language users can focus on designing and analyzing their models' workflow or data flow (Shen et al., 2021, p. 3122).

2.5 textX Meta-Language (Compiler-compiler)

textX³ is an Xtext-inspired *meta-language* and tool written in Python to define IDE-independent, lightweight, and extensible DSLs (Dejanović et al., 2017, pp. 1-2). DSL developers can apply textX to design languages for various purposes, such as *extracting data sources*, *supporting IDE features*, or *building Model-Driven Engineering toolchains*. (Dejanović et al., 2017, p. 2). The tool generates a *parser* and *meta-model* of a given source code from the grammar definition. Furthermore, textX can also act as an *interpreter* by creating Python classes with attributes to represent the model elements and their attributes as defined in their source code and the DSL grammar (e.g., see "Interpreting Model" section in Dejanović (n.d.e))

2.5.1 Motivation

DSLs are subject to changes, particularly in the early stages of development when the language designers need to understand and analyze the domain (Dejanović et al., 2017, p. 2). Therefore, Dejanović et al. consider that DSL development tools should be *lightweight* by being *independent of complex environments*. This feature helps integrate the DSL into diverse application contexts and enhances the practices of agile methods and techniques.

Furthermore, considering classical Python parsing tools can generate *parsers* from 1) a grammar specification or 2) the interpretation of a grammar definition (Dejanović et al., 2017, p. 2). However, this task is only a step in developing a language because developers still need to process the *parse tree* to produce a suitable data representation format (e.g., an abstract representation of the model). This concern is alleviated when applying *language workbenches*, which serve as cost-efficient tools to provide full-fledged DSL construction facilities such as compilers, debuggers, or language-agnostic editor generation (Dejanović et al., 2017, p. 2). Nevertheless, *language*

³<http://textx.github.io/textX/3.1/>

workbenches are complex, usually bind to a specific integrated environment, and most are Java-based tools. Therefore, Dejanović et al. developed textX to address the two concerns of classical *parsers* and *language workbenches*.

textX aims to be a *lightweight* and *versatile* tool that offers fast specification of *concrete* and *abstract syntaxes* for *meta-model* definition. Its only dependency is the Arpeggio *Parsing Expression Grammar (PEG) parser* (Dejanović et al., 2016). In *context-free grammars (CFGs)*, expressing ambiguity in the language syntax makes it unnecessarily difficult to express and parse machine-oriented languages. *PEG* provides an alternative, *recognition-based* formal foundation to declare machine-oriented syntax for expressing ambiguity of a language by not introducing ambiguity initially (Ford, 2004, p. 1). The following subsection describes the workflow of textX to explain its features.

2.5.2 Features

Figure 2.13 describes the workflow of textX from (a) creating a *meta-model* and *parser* from grammar, (b) parsing a model source code to generate its in-memory model, to (c) interpreting the in-memory model.

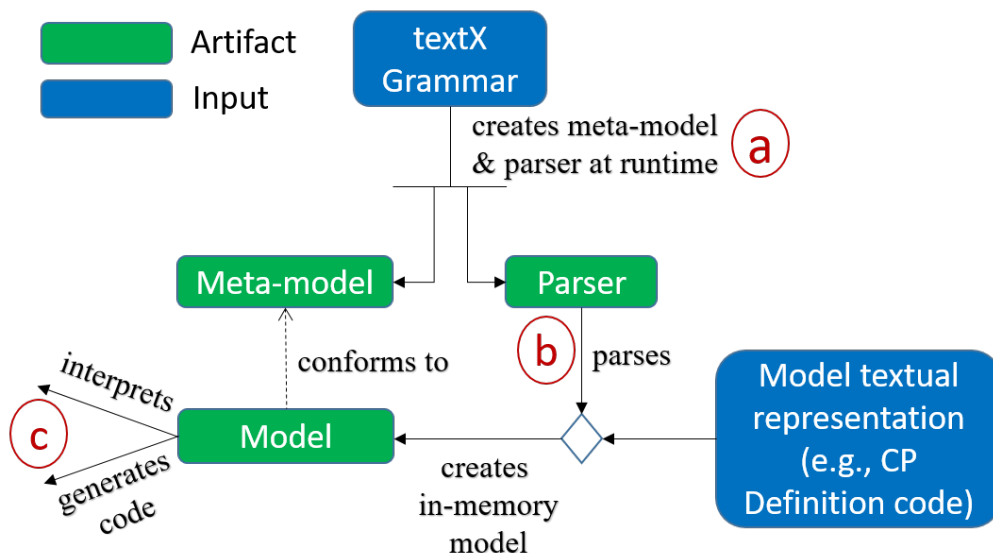


Figure 2.13: Simplified textX architecture and workflow from a) creating *parser* and *meta-model* from grammar, b) parsing the model definition code to generate a model, and c) interpreting or generating code. (Dejanović et al., 2017, p. 3).

1. **Meta-model and Parser Creation from Grammar:** a textX grammar comprises rules that 1) describe concepts of the DSL as *meta-classes* and 2) express the *abstract syntax* of those concepts using textual *concrete syntax*. textX generates a *meta-model* with a hierarchical order consisting of Python classes and attributes to represent the grammar rules. For example, steps (a) and (b) in Figure 2.14 illustrate the grammar of a Robot Instruction language and its generated *meta-model*. The language defines the movement directions and walking steps from the initial position.

Since concepts in grammar can contain their attributes and dependency on other concepts, textX can construct the Arpeggio *parser* from the structure of the concepts. The parser is a graph of Python classes that inherits `ParsingExpression` class (Dejanović et al., 2016, p.

2 Foundations

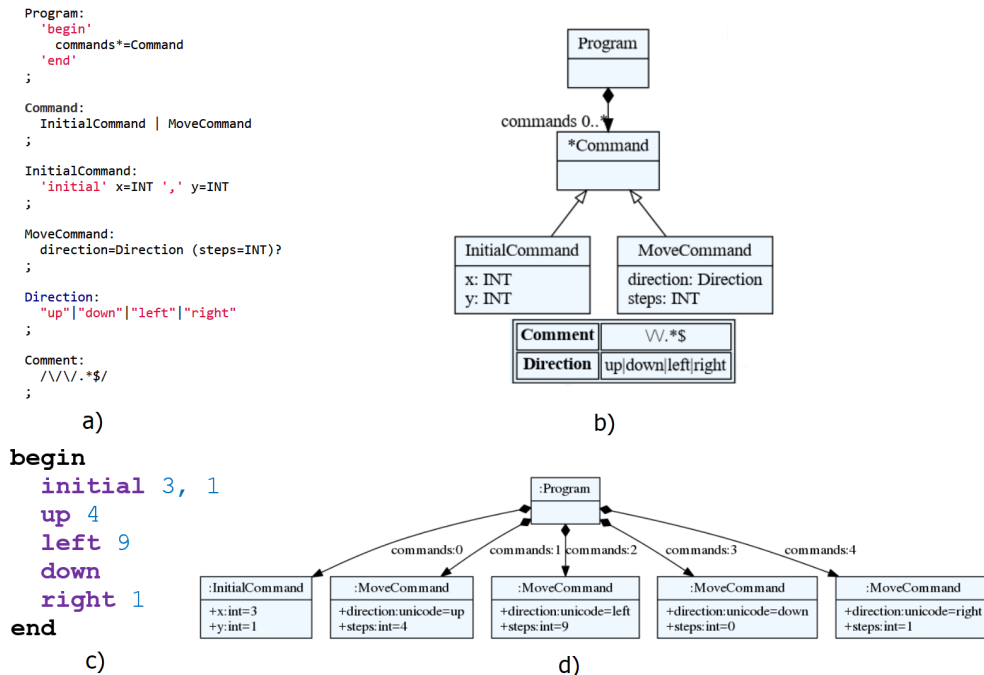


Figure 2.14: textX example workflow for processing a Robot Instruction Language. a) language grammar specification, b) *meta-model* generated from the grammar, c) the source code of a Robot Instruction set, d) the generated model (Dejanović, n.d.e).

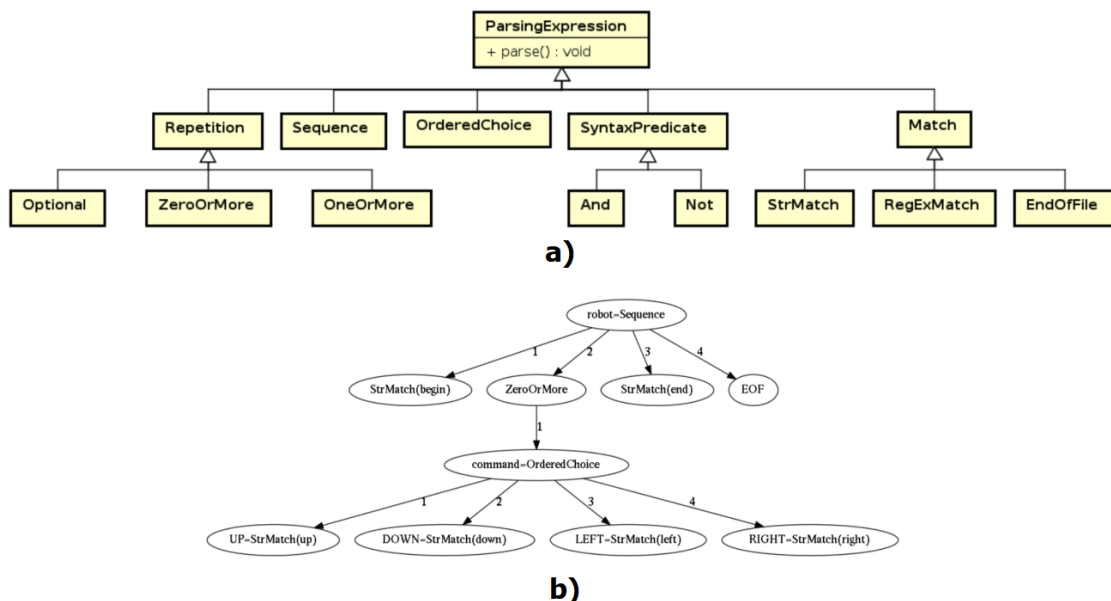


Figure 2.15: Arpeggio ParsingExpression class and example. a) The hierarchy of PEG's classes in Arpeggio, b) The *parser* model of the Robot Instruction Language (Dejanović et al., 2016, p. 3).

2). Since Arpeggio is a *PEG parser*, which is a *recognition-based* system, the ParsingExpression class defines eligible *rules* or *string predicates* in a *meta-language*' (Ford, 2004, pp. 1-2). Figure 2.15 illustrates (a) the hierarchy of Arpeggio ParsingExpression class and (b) the Arpeggio parser of the Robot grammar (Dejanović et al., 2016, p. 3).

2. **Parsing Model:** the generated *parser* in step (a) of Figure 2.13 reads an input source code, which is the DSL's textual representation of the model, thereby creating a graph of Python classes at runtime to express the concepts in the input model (Dejanović et al., 2017, p. 3). Each class in the model is an instance of the corresponding class in the *meta-model*. Because this *meta-model* contains syntax rules (e.g., at least one element exists), it verifies if the structure of the input model is valid and throws a corresponding syntax error in case of constraint violation (Dejanović et al., 2017, p. 4). Step (c) and (d) in Figure 2.14 shows how textX generates the model from a source code of the Robot Instruction language.
3. **Interpreting Model:** The model contains a graph of Python classes and attributes of the concepts defined from the source code. Therefore, a program can access any concept and its elements by traversing the model's *parse tree*, thereby analyzing the extracted data to execute a particular semantic expression (e.g., adding two numbers). For example, figure 2.16 demonstrates how to use Python to extract concepts of the model (d) in Figure 2.14 (Dejanović, n.d.e).

<pre> class Robot(object): def __init__(self): # Initial position is (0,0) self.x = 0 self.y = 0 def __str__(self): return f"Robot position is {self.x}, {self.y}." def interpret(self, model): # model is an instance of Program for c in model.commands: if c.__class__.__name__ == "InitialCommand": print(f"Setting position to: {c.x}, {c.y}") self.x = c.x self.y = c.y else: print(f"Going {c.direction} for {c.steps} step(s).") move = { "up": (0, 1), "down": (0, -1), "left": (-1, 0), "right": (1, 0) }[c.direction] # Calculate new robot position self.x += c.steps * move[0] self.y += c.steps * move[1] print(self) # Execute the robot model from the input source code robot = Robot() robot.interpret(robot_model) </pre>	<pre> Setting position to: 3, 1 Robot position is 3, 1. Going up for 4 step(s). Robot position is 3, 5. Going left for 9 step(s). Robot position is -6, 5. Going down for 0 step(s). Robot position is -6, 5. Going right for 1 step(s). Robot position is -5, 5. </pre>
a)	b)

Figure 2.16: A Python program to interpret the Robot Instruction model using textX facilities.

textX also offers other considerable features for DSL development (Dejanović et al., 2017, p. 3), such as (1) automatic *meta-model* and *parser* construction from a single definition; (2) automatic resolving of model references, enabling the import of external elements (Dejanović, n.d.c); (3) *meta-model* and model visualization, modifiers for (4) repetition expression and (5) grammar rules; (6) case-sensitive/-insensitive parsing; (7) white space handling control; (8) direct support for code comments by treating them as whitespaces; (9) object and model post-processing; (10) grammar modularization; (11) support for debugging and error report.

3 Related Work

The approach of modeling CPs with DSLs can construct effective and efficient digital treatment processes. Therefore, CP modeling witnesses numerous studies proposing different techniques to develop DSLs. One method to classify the implementation of DSLs is to consider their representation. From this perspective, the category of DSLs for CPs are 1) graphical, 2) textual, and 3) a combination of both. This chapter sequentially summarizes state-of-the-art solutions in each category with their supported CP concepts, advantages, points of improvement, and the research gap that our study explores. Table 5.1 in Section 5.1 summarizes the modelable elements in SACM and the DSLs discussed in this section.

3.1 Graphical Domain-Specific Modeling Languages

3.1.1 DSML4CPs - Modeling Clinical Pathways in Oncology Using Extended MEMO OrgML Process Modeling Language

Motivation Considering existing DSMLs focused on executing workflow activities but neglected financial, administrative, and human resource management concepts, Heß et al. (2015) developed DSML4CPs to model CPs, Health Information System (HIS) aspects, and other hospital activities. Furthermore, the authors were concerned about using simple DSMLs with generic semantics details. Although these languages relieve the users of learning and applying CP modeling concepts, they do not provide comparable understandability compared to DSMLs with specialized terminologies and abstractions (Frank, 2010). Hence, the authors designed DSML4CPs with terminologies specific to model Oncology treatment activities, foster communications among stakeholders, and enable model analysis.

Method The authors follow the language design method of Frank (2010, 2013), which consists of seven steps: (1) define scope, (2) analyze requirements, (3) derive specific requirements with a collection of scenarios, (4) specify the language rules (abstract syntax), (5) formulate graphical notations of the language concepts (concrete syntax), (6) develop optional features, and (7) iteratively evaluate and refine the developed artifacts.

Regarding the implementation, DSML4CPs leverages MEMO OrgML, a process modeling language that supports extensions of domain concepts. The authors exert this capability to integrate oncological treatment terminology, syntactic and semantic rules into the meta-model definition. As a result, DSML4CPs can express CPs using OrgML graphical notations with internal constraint validation capability. Figure 3.1 shows a Soft Tissue Sarcoma CP expressed using extended concepts of DSML4CPs.

Strength A remarkable capability of DSML4CPs is the expression of HIS aspects, such as *medical, administrative, human, and financial resources*. These elements are necessary to supply

3 Related Work

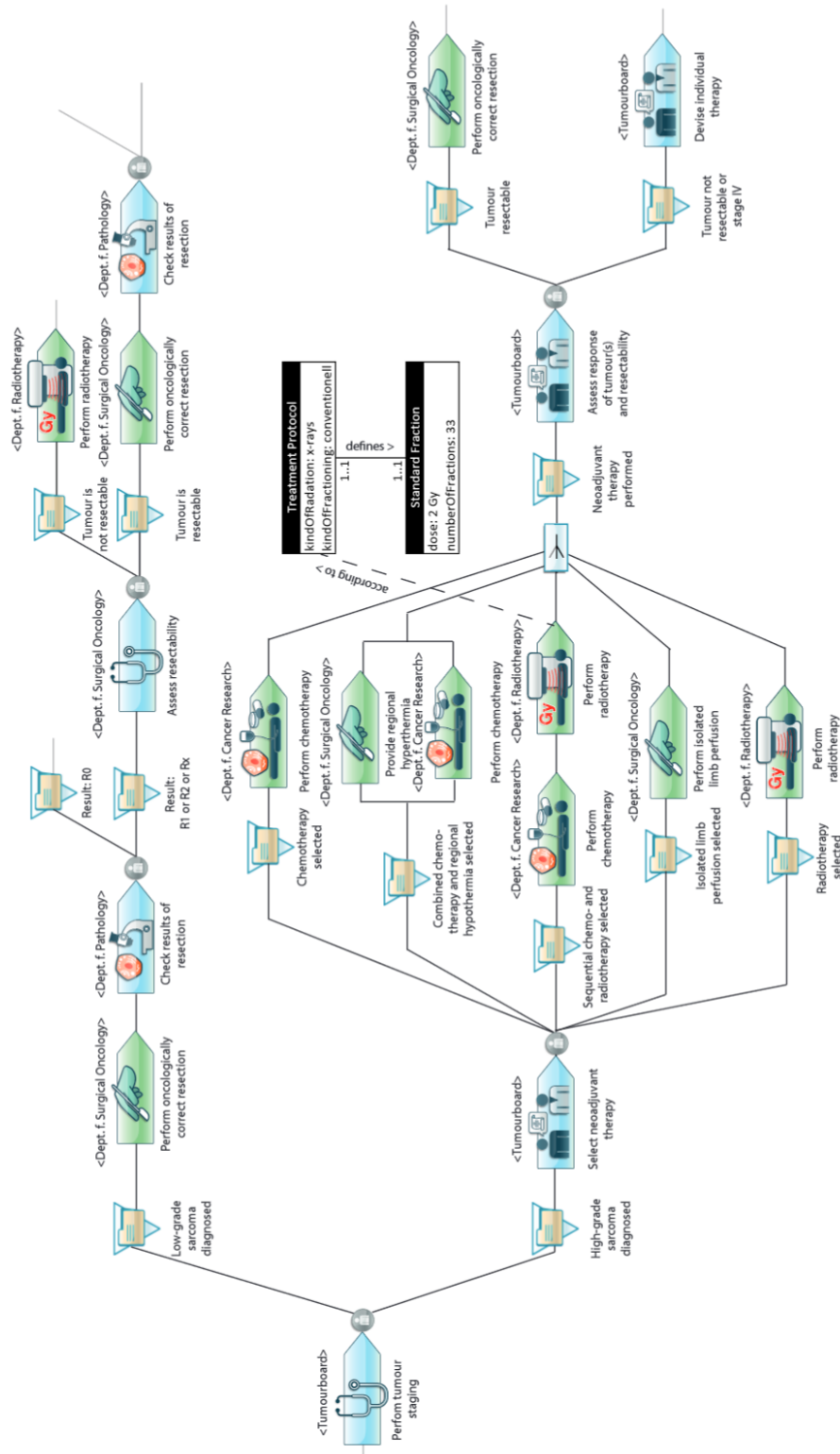


Figure 3.1: Excerpt of a Soft Tissue Sarcoma CP expressed in DSML4CPs (Heß et al., 2015, p. 12).

resources for the execution of healthcare services besides *workflow*, *control flow*, and *responsibility assignment*. Furthermore, MEMO OrgML enables extensions to the DSML. Therefore, defining new terminologies relevant to the treatment requirements is possible.

3 Related Work

Additionally, DSML4CPs offers an intuitive representation of CP elements. The authors adjusted the original graphical notations of MEMO OrgML to express diverse treatment activities (e.g., laboratory or scanning test, diagnosis, surgery). Consequently, the instinctive icons of the workflow elements effectively convey the activity type. The evaluation result partly unveiled the understandability of the DSML's UI and Oncology-specific concepts, as the stakeholders stated that DSML4CPs fosters communications among involved parties (Heß et al., 2015, p. 13).

Research Gap Heß et al. (2015, p. 4) stated that modeling CPs of other medical domains requires an extension of their professional terminology. Therefore, the ease of use, adaptability, and implementation to model CPs in other medical fields is unexplored. Furthermore, e-Health applications may need to communicate with partner systems for data integration or synchronization, yet DSML4CPs does not support this capability. Based on these unexplored potentials, Acadela aims to discover 1) the ability to model CPs in different medical fields as a generic DSML and 2) support communication to external systems.

3.1.2 BPMN4CP - Modeling Clinical Pathways by Extending BPMN

Motivation Realizing the original BPMN framework cannot fully express unpredictability in a treatment process, Braun et al. (2014, 2016) extended BPMN to include capabilities to model parallel flow, variable flow, and necessary elements for evidence-based decisions. Besides, the DSML can also model clinical documents and resources (e.g., Human, Medicine, Room, Transportation) management capabilities (Braun et al., 2016). Figure 3.2 illustrates an example BPMN4CP model of a (simplified) stroke CP.

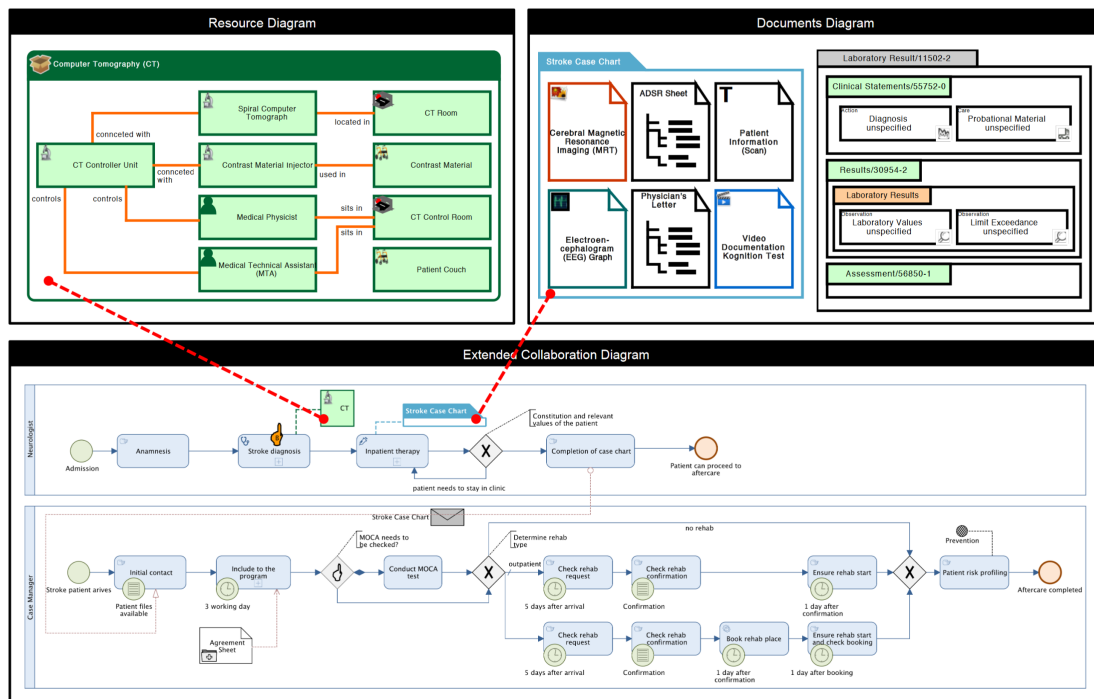


Figure 3.2: BPMN4CP model of a (simplified) stroke CP with workflow, resources and documents (Braun et al., 2016, p. 29).

3 Related Work

The authors leverage the BPMN's kernel because of the following benefits:

1. Promote tool integration, model exchangeability, and unambiguous language interpretation with its meta-model definition
2. Extensible thanks to a structured definition and lightweight mechanism
3. Recognized by the industry and academia world
4. Qualified as an official ISO standard (International Organization for Standardization, 2013)

Method Braun et al. leverage BPMN "extension by addition" mechanism (Stropi et al., 2011, p. 3) to add domain-specific concepts into BPMN. Each *ExtensionDefinition* has *ExtensionAttributeDefinition* to define the attributes of an *Extension* concept. Each *ExtensionAttributeDefinition* contains *ExtensionAttributeValueDefinition* to dictate the (primitive) data type of an attribute. The final step is to bind the *Extension* element and attributes into BPMN using its *ExtensionDefinition*.

To integrate the extended domain-specific concepts systematically, the authors adapted the model-transformation-based procedure model of Stropi et al. (2011, p. 5) to formulate a 3-step process (Braun et al., 2014, p. 2):

1. **Domain Analysis and Equivalence Check:** Verifying whether an extended concept is representable by any equivalent BPMN elements. If the semantics of the domain-specific concept is not expressible using the default BPMN elements, then create a custom *Extension Concept* element. Otherwise, define the extended concept as a *BPMN Concept*.
2. **Modeling the Domain:** Define the Conceptual Domain Model of Extension (CDME) as a UML class diagram to express all the *Extension Concepts* and *BPMN Concepts* defined in Step 1.
3. **Transform the CDME to a BPMN Extension Model:** Deriving the BPMN+X from the CDME configuration. BPMN+X is a UML profile consisting of stereotypes (e.g., *BPMNElement*, *ExtensionElement*, *ExtensionDefinition*) to express the extended concepts in BPMN.

Strength BPMN4CP supports various documents and resource management capabilities. Thus, medical experts can have an overarching view of treatment activities, required supplies, and relevant documents. Besides, the language can flexibly extend new medical concepts by defining extra *Extension* objects. This feature enables the inclusion of domain concepts from different medical fields to model their CPs. Reusability is another advantage, as it can export or import BPMN-based medical conceptual models; hence, BPMN4CP can benefit from or export compatible CP models to other BPMN-based DSMLs.

Research Gap Regarding functionality, BPMN4CP does not support communication with external systems or importing CP elements. Additionally, from the demonstrated CPs of the extended BPMN4CP, the capability to visualize medical data and process backward variable flow can be possible. However, the example CP models did not present this feature. Finally, the opinion of modelers and medical experts regarding the usability and accuracy of the treatment process is unexplored. Meanwhile, our study incorporates the feedback of the users to discover the applicability of Acadela.

3.1.3 BPMN^{SIX} - Modeling Surgical Workflow by Extending BPMN

Motivation BPMN^{SIX} aims to model the *workflow*, involved *medical resources*, and *system operability* for CPs of integrated operating rooms (ORs). (Neumann et al., 2016, 2017)

Method Neumann et al. applied the procedure of Braun et al. (2016) to define and integrate extended OR concepts into BPMN in six steps: 1) *analyze requirements* based on a domain-specific use case and literature review, 2) *investigate different modeling languages* to determine their applicability in modeling CPs, 3) *perform equivalence check* to identify *Extensions* to the original BPMN notations, 4) *define a CDME* as UML Diagram to 5) *create an abstract syntax* by transforming the CDME into a valid BPMN extension model, and 6) *define the concrete syntax* with graphical notations for the extended concepts (Neumann et al., 2016, p. 3).

The OR concepts which BPMN^{SIX} models are *Surgical Activities*, which includes *Actuator* (role), *Used Body Part*, *Surgical Action*, *Anatomical Structure*, and *Used Resource or Instrument*; *Surgical Phase* composes of *Surgical Activities*; *Intervention* is the lowest granularity level that comprises *Surgical Phases*; *Process Flow* covers 1) *Parallel Flow* execution of simultaneous tasks, 2) *Exceptional Treatment* contains irregularities and complications in the treatment process, and 3) *Resource Exception* specifies resource unavailability (Neumann et al., 2016, pp. 3-4). Figure 3.3 demonstrates a cataract surgery CP expressed in BPMN^{SIX} (Neumann et al., 2017).

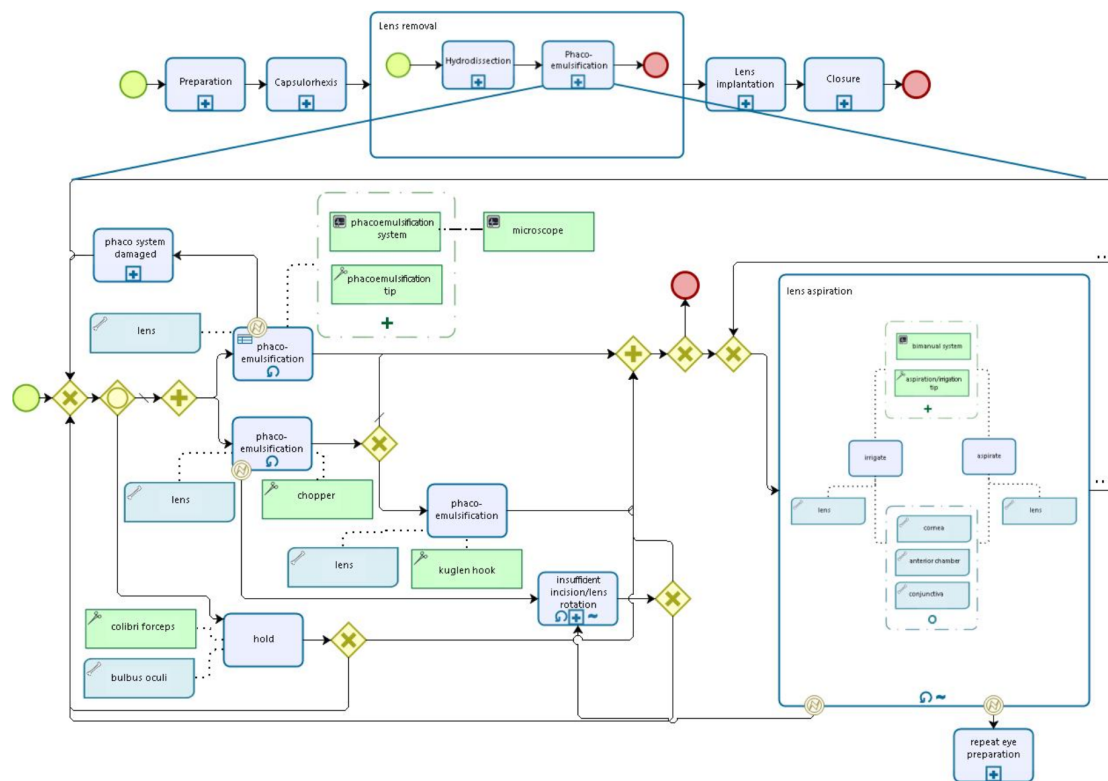


Figure 3.3: BPMN^{SIX} model of a cataract surgery process and a part of *Phacoemulsification* subprocess

Strength BPMN^{SIX} has similar advantages to BPMN4CP, which enables the definition and reusability of future domain concept extensions. In addition, the DSML demonstrated its capability to model backward variable flow. In conclusion, BPMN^{SIX} can model relevant aspects from workflow activities to resource management in CPs for ORs.

Research Gap Similar to BPMN, BPMN^{SIX} did not explore the usability of the language from the modelers' perspective. Additionally, external communication and importing CP elements are not mentioned in the features of BPMN^{SIX}.

3.2 Textual Domain-Specific Language

3.2.1 FCIG - Modeling Clinical Guidelines using Xtext

Motivation Realizing Electronic Medical Records (EMRs) depends on clinical decision support guidelines to improve care quality, reduce medical errors and facilitate decision-making processes. Msosa (2018, 2019) developed a framework named FCIG to model and maintain *Clinical Practice Guidelines* (CPGs) in the form of *Computer-interpretable Guidelines* (CIGs). The CIGs aim to support clinical information systems of low- and middle-income countries. The study focused on modeling two CPGs adapted by the World Health Organization (WHO), namely Integrated Guidelines of the Management of HIV and Integrated Management of Child Illnesses.

Method First, the author identifies the semantics for CIG elements. Specifically, each CIG is a *Guideline* containing a list of *Recommendations*. Each *Recommendation* has a list of *Conditions* and *Actions*. A *Condition* has a *DecisionVariable* (type String), *Relator* (e.g., "is", <, ≥), and *VariableValue*, which is a number, String, or boolean value. An example of a *Condition* is "HIV rapid test result is negative". An *Action* has an *ActionVerb* to denote a function and an *ActionVerbComplement* to specify the object. For instance, in the "Prescribe regimen 4" *Action*, "Prescribe" is the *Action Verb*, and "regimen 4" is the *ActionVerbComplement*. The author then built the abstract syntax from the above semantics.

To implement the concrete syntax, Msosa (2018, p. 73) leveraged the *Xtext*¹ framework to build grammar rules for FCIG. In addition, the DSL has an IDE to support modelers in defining CIG elements. Figure 3.4 shows an example of the HIV CIGs model expressed in FCIG.

For the evaluation of FCIG, the author invited 6 experienced and 13 novice CIG modelers and presented the three primary grammar constructs (*Recommendation*, *Condition*, *Action*). Then the participant followed a link to evaluate the language by completing a System Usability Scale (SUS) Questionnaire. The SUS mean scores of the experience and novice modelers are 89.17 and 79.23, respectively.

Strength The SUS score implies FCIG is user-friendly to both novice and expert modelers. From this observation, the language syntax and IDE can potentially increase productivity and user experience. The reason is that FCIG is learnable to both experts and apprentices. When combined with IDE features such as syntax highlighting or auto-completion, the modelers can efficiently construct CIGs.

Another advantage of FCIG is its ability to model alternative treatment paths thanks to the conditional activation of medical interventions. This ability supports parallel flows, backward and forward variable flows as multiple simultaneous activities can be repeated whenever their conditions are satisfied.

¹Xtext: <https://www.eclipse.org/Xtext/>

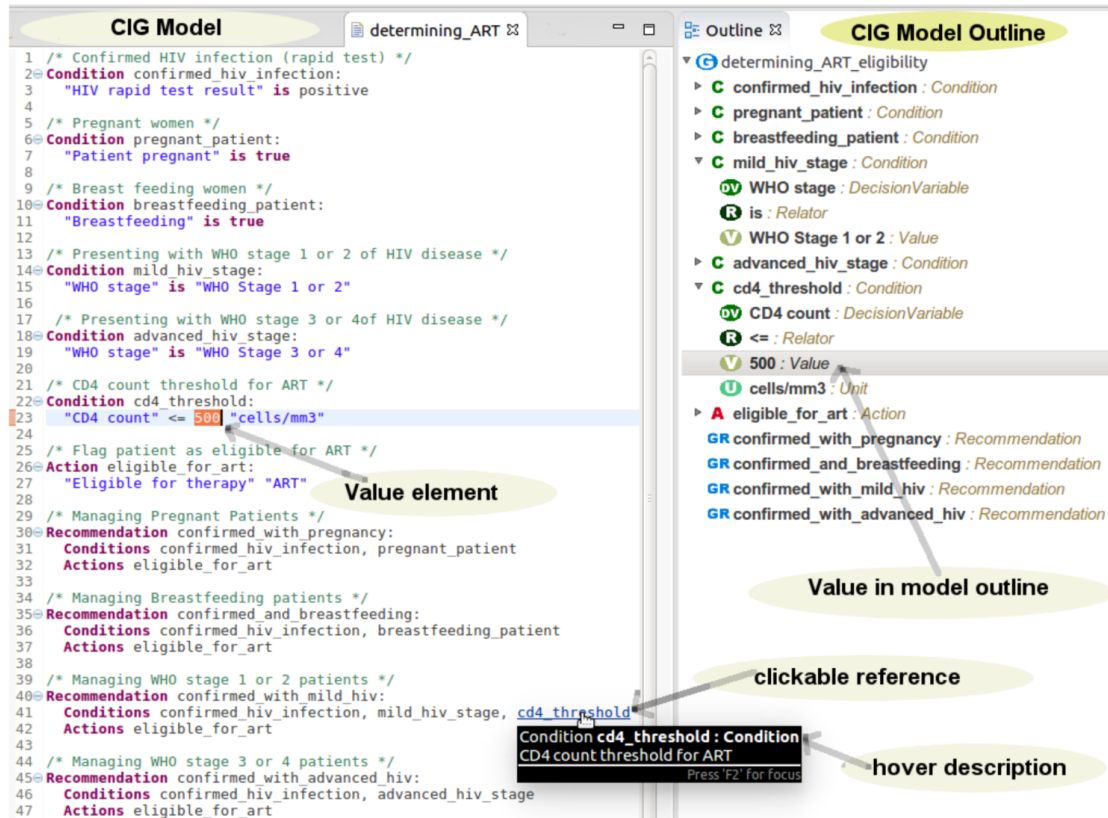


Figure 3.4: A CIG model (left) and its outline (right) in FCIG IDE

Since *Xtext* can express FCIG models in the Ecore meta-model of Eclipse EMF², the CIGs models of the DSL are reusable in a different system that utilizes Ecore to construct CIGs.

Research Gap Regarding the language functionality, the ability to send requests to external systems is unexplored in FCIG. For example, the language may need a different construct to request APIs if a clinical guideline needs medical guidance from another service. In addition, a difference between Acadela and FCIG is that Acadela needs to support the visualization of medical data, which is not in the scope of FCIG. Furthermore, Acadela has a constraint validator to help users identify and fix syntax and semantic errors in their code.

Concerning the usability of FCIG, in the DSL evaluation, the modelers participated in a paper-based session to review FCIG grammar. This result demonstrates that the participants can understand the language, yet it does not expose their ability to command and apply FCIG. In Acadela, we examined the usability of modelers by presenting a modeling scenario such that they have to use the IDE features to model missing elements of a CP. This context demonstrates that participants can understand and use the language to execute their modeling tasks.

3.2.2 Prescriptive Grammar for Clinical Describing Workflow

Because the paper is not freely and fully accessible, we could only gain limited insights into the language implementation, strength, and research gap.

²EMF: <https://www.eclipse.org/modeling/emf/>

3 Related Work

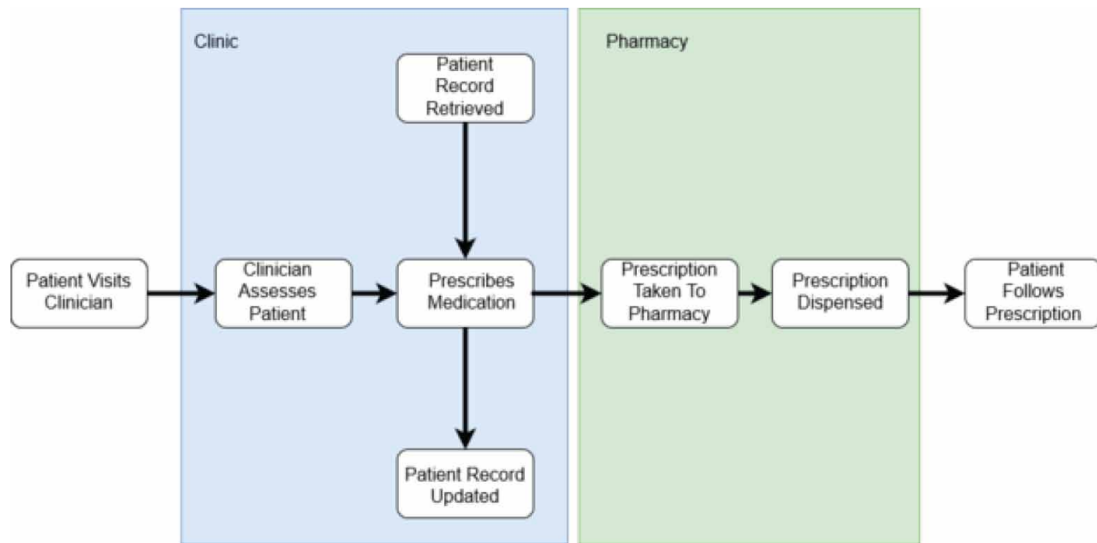


Figure 3.5: A prescription workflow that the DSL models.

Motivation Clinical experts are working on various tools that are cumbersome and not seamlessly integrated into each other, leading to disconnected and inflexible prescription workflows. Therefore, Kauranen et al. (2019) developed a textual DSL to simultaneously model a clinical prescribing process and maintain the Electronic Health Record (EHR).

Method The DSL uses a Parser Combinator Grammar (PCG) to directly and indirectly map medical concepts to a syntax. Additionally, there is a custom IDE for medical experts to create their prescriptions. Finally, the result of the process is stored in the EHR under HL7 standard. Figure 3.5 illustrates the prescription process that the DSL models.

Strength HL7 is a standardized format used in multiple HISs. Hence the model produced by the DSL is potentially compatible or reusable in different e-Health applications. Furthermore, textual DSLs are extensible by defining new sub-grammar to represent extended medical concepts.

Research Gap The ability to model various CPs in different medical fields is unexplored. Additionally, the demonstrated prescription workflow is a linear process. As a result, the possibility of modeling parallel or repeated activities is not yet illustrated. Furthermore, medical experts may need to consult external services to learn about side effects or chemical interactions among medicines. However, it is unclear if the language supports this capability. Meanwhile, Acadela provides constructs to define the CP data, triggered event, HTTP method, and response error message for external requests.

4

Smart Adaptive Case Management (SACM)

Integrated care is a widely recognized approach to providing patient-centered treatment. However, medical facilities need more effective software solutions to address concerns in practicing integrated care. Alternatives to using e-Health systems are paper-based methods that hinder effective collaborations among professionals; meanwhile, hardwiring a workflow engine in a frontend environment requires a remarkable effort to evolve the model for handling treatment variations continuously (Michel, 2020, p. 3). Therefore, Felix Michel (2020) developed SACM as a meta-model-based ACM for Integrated Care (ACM4IC) that incorporates case-based collaboration, adaptive modeling, and customization of treatments.

In our study, SACM is the e-Health application that provides CP definition, execution, and management features. Therefore, Acadela demonstrates its potential to model CPs in e-Health systems by compiling CP definitions in Acadela to a SACM format, such that SACM executes the CPs having the expected behaviors. In other words, CPs defined in SACM function correctly as CPs defined in Acadela. However, Acadela implements several optimization features to enhance the CP modeling process.

First, SACM defines meta-models of the treatment templates using XML, which contains reducible elements (e.g., opening and closing tags, declaration of the default path reference that can be automatically inferred). Furthermore, the XML templates are not modularizable, i.e., they cannot import an element from an external XML file. For these reasons, Acadela aims to optimize the definition of treatment meta-models with a more concise, reusable DSL that is user-friendly and learnable to modelers. In parallel, Acadela also visualizes the treatment pathway as a shared artifact to foster communication between technical and medical experts.

4.1 Problem Description

Integrated care is a practice that promotes multidisciplinary medical professionals, patients, and carers to collaborate in providing patient-centric treatments using integrated technological solutions. Additionally, integrated care empowers patients for self-management and continuously monitors their activities to propose the corresponding recommendations and suggestions (Vargiu et al., 2017). Operating an integrated care environment requires the system design to address multidimensional challenges; the following three concerns are significant and prevalent difficulties in developing integrated care systems:

1. **Highly Context-dependent, Unpredictable Treatments:** Hollingsworth (2010) states that interactions among medical conditions, medications, and treatments result in indeterministic care procedures. Furthermore, due to interruptions, unpredictability, and tight collaborations, treatment paths are *context-dependent* and do not have predefined strategies to achieve the care goal (Horsky et al., 2005). Additionally, medical professionals need to include the latest medical knowledge or evidence-based practices to handle variations in

4 Smart Adaptive Case Management (SACM)

the treatment process (Garde & Knaup, 2006). Therefore, medical facilities require a versatile system that can efficiently adapt treatment executions based on the patient case's context.

2. **System Interoperability and Semantic Information Exchange:** Integrated care system involves a diverse range of healthcare services, leading to a complex system design. The intricate system landscape poses challenges in coordinating the services, i.e., sharing information necessary for an efficient healthcare operation. Furthermore, progressively specializing in disease-focused medicine contributes to service fragmentation which jeopardizes holistic care principles (Valentijn et al., 2013).
3. **Coordination across Multiple Organizations and Different Roles:** Patient treatment is a knowledge-intensive process that usually involves the coordination of multidisciplinary professionals from diverse organizations. Overlooking the communication among experts can increase health resource consumption and might adversely affect the treatment outcome of patients (Garde & Knaup, 2006). In contrast, precise and immediate communication is a critical factor in delivering treatments (Horsky et al., 2005).

Figure 4.1 demonstrates the collaboration problem in an integrated care environment without integrated tool support on the left and with integrated tool support for care professionals on the right. Depending on the healthcare system, medical professionals can be distributed across multiple organizations (e.g., in Spain) or be concentrated in one organization (e.g., Israel). Given that medical treatments usually record patient conditions in questionnaires, uncoordinated documentation of patient data can lead to redundant evaluations. Consequently, the care process contains slightly different results, hence drawing diverse conclusions for the case. Furthermore, uncoordinated execution of concurrent therapies in chronic disease treatments can lead to undesirable side effects. Therefore, integrated care must exchange semantic information across organizations and foster collaboration among healthcare professionals to enable contextualized and patient-oriented treatment.

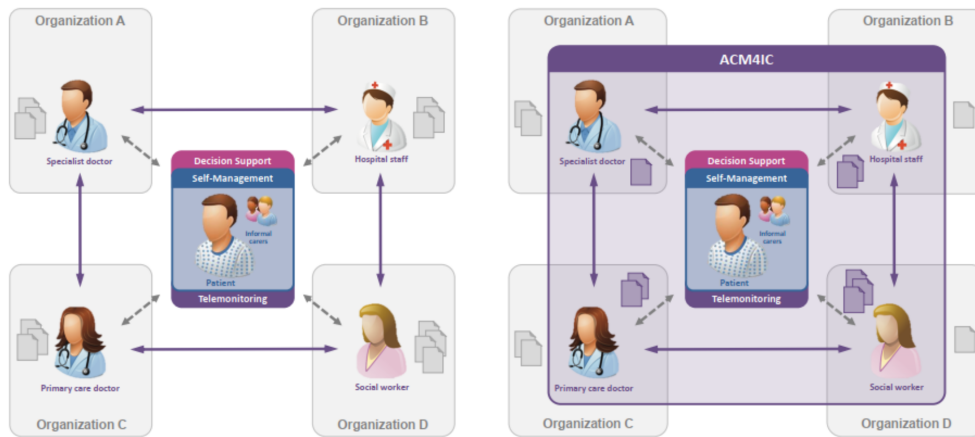


Figure 4.1: Visualization of the problem in the Integrated Care Environment without integrated tool support (left). Each organization stores medical data independently, leading to redundancy and uncoordinated analysis of the patient's medical status, potentially resulting in undesirable outcomes. An integrated care environment empowered by integrated tools (right) can offer consistent documentation of critical data, thus fostering collaboration and communication among care professionals throughout the care process. (Michel, 2020, p. 3)

To address the above challenges, Felix Michel develops SACM with the following features:

1. **Adaptive Case Management:** One core principle of ACM is promoting knowledge workers to apply their knowledge and experience to handle changing circumstances (Swenson & Palmer, 2010). This doctrine is particularly suitable for medical treatments as medical professionals can adapt the treatment activities to tackle unprecedented variants of the treatment process. SACM supports this ability by defining treatment-specific case templates containing mandatory and optional care activities. Therefore, medical professionals can adaptively conduct patient-centric treatments at runtime, i.e., selecting the necessary treatment tasks depending on the medical condition of an individual patient or the occurrence of unpredictable situations.
2. **Purely Meta-model-based Integration Patterns:** SACM enables the configurations of all system layers by modifying the corresponding meta models (Michel, 2020, p. 33). In other words, developers change data schemata, process models, discretionary access rights, and user interface models by adjusting the meta-model of the case. This feature offers flexibility in adapting to the desired needs of the treatment as developers do not have to modify any application code; but rather adapt the meta-models accordingly.
3. **Incorporation of Case-based Collaboration Capabilities:** Integrated care treatments require coordinating medical professionals across different medical departments to manage various aspects of the care process. For this reason, SACM offers communication features to support medical experts in 1) recording data, 2) exchanging case information, 3) assigning tasks based on the roles of the involved medical experts, 4) receiving notification of any adversary or critical medical status, and 5) summarizing critical medical data of the patient case (Michel, 2020, pp. 38-39).

The following subsection further describes the specific requirements and motivations to realize the three features of SACM.

4.2 Requirements

Based on literature research, Michel (2020, p. 33) designs SACM to conform to the following non-functional requirements:

1. **Operating as a Software as a Service (SaaS) System** that supports multi-tenancy.
2. **Applying Container-based Deployment** using Docker to provide a reliable and fast deployment process (Bernstein, 2014, pp. 82-83).
3. **Providing API Interfaces** to support system interoperability and extendability.

Additionally, Michel (2020, pp. 33-39) identifies three high-level functional requirements to develop an ACM software system. This section presents the decomposition of these requirements.

4.2.1 R1: Support a Purely Meta-Model-Based Approach

Innovative ACM requires a swift and flexible case configuration to address treatment evolution promptly. Therefore, modeling case templates as meta-models is a potential approach because meta-model-based systems enable adaptive and consistent case modification to the needs of medical experts without changing the application code (Matthias, 2011). In other words, case elements such as data schemata, workflow model, discretionary access rights, and appearance of case items are directly configurable by updating their attributes in the case meta-model. Upon its

4 Smart Adaptive Case Management (SACM)

completion, the modified case meta-model is ready to use without redeploying the system. As a result, modelers and care professionals can efficiently collaborate to create or adapt case templates and operate them afterward. To realize this benefit, a meta-model in SACM shall support the definition of the following models:

R1.1: Data Schema Models

Requirement: *SACM must model the data generated during the case execution or data returned from an interaction with a third-party system. An example of the latter is a patient profile and medical conditions returned from a hospital information system. (Michel, 2020, p. 34)*

Motivation: Case Management integrates data into process executions (Hauder et al., 2014a, p. 99). To support this characteristic, The CMMN CaseFileItem element stores case data directly inside the case or as references to information sources (Object Management Group, 2016, p. 22). Those referable data sources can be objects or files that encapsulate data. This feature enables reusable and nested data structures, as one can define the reference to a parent or child data source (Object Management Group, 2016, pp. 22-23). The data stored in these referable objects can be structured or unstructured, simple or complex, such as key-value pairs or unstructured XML documents. To model elements in integrated care treatment, SACM leverages the CMMN concepts and nested, reusable data structure while adding extensions to model concepts not defined in CMMN, e.g., HTTPHook for sending HTTP requests to external services.

R1.2: Adaptive Process Models

Requirement: *The system shall define adaptive treatment templates that are customizable to the requirements of hospitals or specific treatments. The system should define care processes using Adaptive Case Management as the reference methodology. These processes shall be synchronizable to other systems for supporting integrated care. (Michel, 2020, p. 34)*

Motivation: Clinical processes comprise both structured, predefined procedures and indeterministic treatment paths. Meanwhile, the essential capability of ACM is managing processes containing both unpredictable execution paths and deterministic business workflows. ACM possesses this ability by incorporating the structuring concepts of traditional workflows into its mechanism (Burns, 2011). Figure 4.2 illustrates the four different categories of processes based on their degree of structure.

According to the classification method, *structured processes* do not permit exceptional execution paths during the process. A *Structured process with Ad-hoc exceptions* and *unstructured processes with predefined fragments* are typical for ACM because they allow alternative execution paths to handle unpredictable situations. Meanwhile, *unstructured processes* execute activities spontaneously. In general, processes exchange a *high degree of flexibility* and *knowledge intensity* with a *lower degree of automation* and *predictability*, and vice versa.

CMMN supports modeling this ACM behavior of deviating from a structured process or selecting suitable activities thanks to the Sentry element (See Section 2.3.2). A Sentry activates Stage(s) or Task(s) based on a conditional expression. Since CMMN Stage can contain both structured process fragments with predefined workflow and optional activities, a Sentry applied to a Stage

4 Smart Adaptive Case Management (SACM)

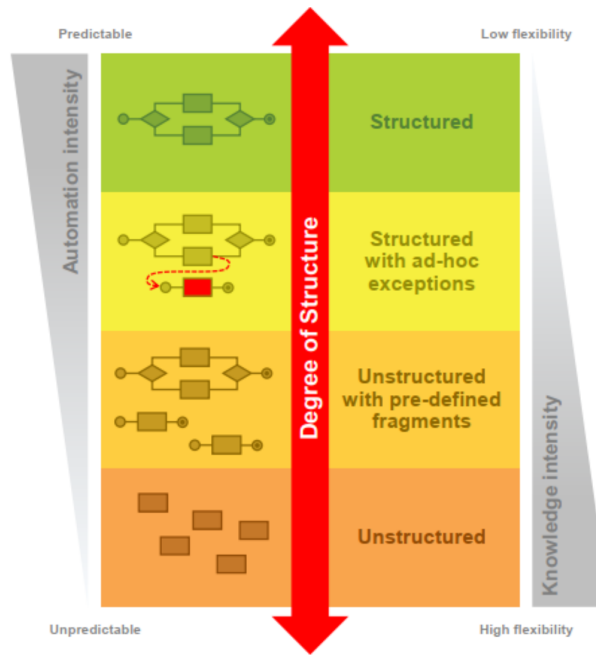


Figure 4.2: Degree of process structure according to Michel et al. (adapted from (Di Ciccio et al., 2012))

can trigger an exceptional execution path. Additionally, modelers can construct unstructured processes using the Discretionary Stage/Task. Applying CMMN concepts of Sentry, Discretionary Stage/Task, and Plan Stage/Task enables runtime planning (Kurz et al., 2015, p. 4), which is essential for handling unpredictable circumstances (Hauder et al., 2014a, p. 104; Kurz et al., 2015, p. 3).

R1.3: Support Role-Based and Discretionary Access Right Model

Requirements: *The system needs to support granular role-based and discretionary access control mechanisms to enforce who can access which patient cases. Additionally, the system must assign clinical tasks based on user roles. (Michel, 2020, p. 35)*

Motivation: Executing each patient case in integrated care pathways involves the collaboration of *multidisciplinary care professionals* (White, 2009). However, each care team member may have a *different level of involvement* (Burns, 2011). Therefore, processes shall support defining responsibilities of each professional, such that they can 1) execute designated medical interventions and 2) access authorized sensitive medical data of the patient while ensuring data privacy (Michel, 2020, p. 35).

A crucial factor for role-based and discretionary access control is **transparent responsibility**. This characteristic specifies the designated authority of all caseworkers, hence facilitating the collaboration among them (Herrmann & Kurz, 2011; Kurz, 2013, pp. 89-90; Hauder et al., 2014b, p. 26). Specifically, authorized case workers can properly assign others to execute a new activity as they know the other users' roles. Therefore, the system shall support granting access to case activities **dynamically at runtime**. As a result, knowledge workers can influence and adapt various case aspects at their discretion and perspective within their designated authority (Kurz, 2013, p. 90).

Regarding the access control, the system shall support the definition of *read*, *write*, and *owner* access level for each responsible care professional of the case.

R1.4 Support Simple User Interface Models

Requirements: *The user interface must represent each model element using a generic model-based approach while supporting simple layout designs to display tasks. Furthermore, "the default representation must be overridable with a declared special representation to support dedicated clinical use cases and ensure extendability" (Michel, 2020, p. 36).*

Motivation: Tractinsky et al. (2000, pp. 139-140) discovered strong correlations between the user's *aesthetic perception* of an interface and their *usability perception* of a system. In other words, a beautiful user interface is an indispensable factor in engaging users to interact with the system. Furthermore, *software quality* also remarkably affects the *usability* and *acceptability* of the system (Suduc et al., 2010, p. 145). To enhance the quality, the system shall support handling variations in clinical processes, which requires incorporating exceptional elements into the representations. Therefore, it is crucial to support display customization to reflect deviation from the default representation. This feature minimizes the efforts to design and manage multiple static layout designs. In addition, modelers can adapt the representation to display data or visual elements generated by the deviated execution. The adapted visualization helps medical experts view both the default and exceptional data to analyze and handle the case.

4.2.2 R2. Integration with External Services

Patient-centric treatments in integrated care mainly depend on aggregated information (Michel, 2020, p. 36). In this context, non-integrated information sources interrupt the work of knowledge workers (care professionals) and postpone complying activities (Matthias, 2010). Therefore, SACM should 1) support the *integration of external data sources* to prevent duplicate data and allow access to legacy or shared information and 2) *orchestrate processes in third-party systems* to incorporate or trigger external events during the treatment execution.

R2.1: Support External User Identity Management

Requirements: *The system shall provide a Single Sign-On to authenticate care professionals. Therefore, the system shall support external user identity management. Furthermore, the system should persist foreign identifiers of external data as internal primary keys to simplify the integration.*

Motivation: The microservice architecture is a widely used strategy to address the separation of concerns. Applying microservices to the medical context requires most services to authenticate and authorize users. This access control mechanism is necessary to govern the interactions with confidential patient data (Michel, 2020, p. 37). A centralized user identity management supports the realization of 1) a consistent authentication policy across microservices and 2) a Single Sign-On strategy that enables users to authenticate once with a central authentication authority; thus, they can access permitted and protected resources afterward without re-authenticating

4 Smart Adaptive Case Management (SACM)

(De Clercq, 2002, pp. 40-42). A successful login to the central authentication authority typically generates *encrypted tokens*, which are returned to the client and included in subsequent requests to microservices (Bánáti et al., 2018, p. 1183). The case execution engine of the system should manage the case authorization because the authorization policy primarily depends on the instantiated case.

R2.2: Support Process Orchestration of Third-Party Systems

Requirement: *The system shall support the orchestration of external systems to provide integrated care services. Therefore, SACM shall seamlessly integrate with external system processes (e.g., patient self-management process) to provide aggregated process information for care professionals.*

Motivation: Organizations possess legacy systems that are crucial in any offered solutions. However, core aspects of case management that require interaction with the legacy systems have a low degree of automation (White, 2009, p. 11; Kurz et al., 2015, p. 4). Therefore, seamless integration with existing systems is necessary for effective case management (White, 2009, p. 11). Furthermore, case management needs to support external events affecting the case's decision-making. For example, an external service notifies a case management system when the monitoring data of a patient exceed a certain threshold. Therefore, ACM systems require an integrated process orchestration for interoperability with existing systems (Kurz et al., 2015, p. 4).

R2.3: Support Semantic Integration of External Data Sources

Requirement: *Hospital information systems operate using extensive amounts of patient data. Therefore, system architecture shall enhance internal data with external data sources.*

Motivation: One or several integrated HISs typically manage patient information to prevent inputting data multiple times (Michel, 2020). However, information distributed across multiple systems increases the complexity for caseworkers (Matthias, 2010). Case management has a low degree of automation; Nevertheless, it typically requires data from legacy information systems to operate (White, 2009, p. 11). In this case, caseworkers usually copy-paste data from one system to another; or manually transfer documents to their colleagues. Therefore, efficient case management systems should seamlessly coordinate and integrate diverse supporting systems. This ability reduces data duplication and manual handoffs in the current operation and supports the development of future solutions (White, 2009, p. 11).

4.2.3 R3: Support Communication and Coordination

Integrated care typically consists of care teams distributed across different organizational boundaries. Each team comprises multidisciplinary specialists that are responsible for certain aspects of the case treatment. During the case handling, the team documents the results, discusses possible treatments, and coordinates with colleagues to provide the solutions (White, 2009, pp. 4, 11). Since care professionals make decisions based on advice and knowledge exchange (White, 2009, p. 4), active information exchange is vital in managing cases.

R3.1: Support Process Contextual Notifications

Requirements: *Care professionals can prescribe patients specific tasks via a third-party system, e.g., measuring blood pressure every morning. The system should notify the responsible care professionals when the blood pressure exceeds an individually specified threshold. When encountering unforeseen technical situations, the system should also notify professional users.*

Motivation: Knowledge-intensive processes must handle unpredictable circumstances that may require manual exception handling. Since tight information integration is critical for efficient case management (Matthias, 2010), the system must directly link notifications to their triggered process element or task. Only the case owner and care professionals responsible for the process should receive the notification to avoid unnecessary distractions to other caseworkers.

R3.2: Support Direct Case-Based Communication

Requirement: *The system shall support information exchange with case-based professional-to-professional messages; thus, all involved care professionals can follow ongoing conversations. The system shall support direct communication with the patient in a separate conversation to provide integrated care.*

Motivation: Knowledge workers typically use various tools (e.g., communication, workflow management applications) during the case execution (Motahari-Nezhad & Swenson, 2013, p. 264), including email-based applications as an established communication channel; and chat functionality to support case management and collaboration (Motahari-Nezhad & Swenson, 2013, p. 267). Furthermore, the system requires tight integration between authentication and case-based authorization to control access to sensitive patient information in the cases.

R3.3: Support Unstructured Case Notes

Requirement: *The system needs to provide a wiki-based notes area for professionals to document unstructured information collaboratively. Additionally, the system should provide individual predefined templates depending on the case definitions.*

Motivation: ACM shall organize all case-related information efficiently and easily accessible, given that decoupled information may be lost or unavailable when required (McCauley, 2011). Furthermore, the system should enhance flexibility by supporting optional template declaration to customize case notes for dedicated treatments (Michel, 2020, p. 39).

R3.4: Support Case-Template Specific Summaries

Requirement: *The case template must support declaring summary sections to provide a case-specific overview. The summary sections assist professionals in quickly identifying a patient's status based on crucial case information.*

Motivation: Kurz et al. (2015) states a requirement to visualize the case progress to help caseworkers identify case aspects of their designated tasks. According to Michel (2020, p. 39), reporting the case progress using simple generic metrics does not sufficiently represent the *actual state* and *case progress*. Although mathematical metrics may express the percentage of completed tasks, they cannot sufficiently incorporate subsequent activities that are dynamically added during runtime planning. Nevertheless, case workers gain insight into the progress of a case from its existing data (White, 2009, p. 11). Therefore, the system shall support the declaration of case-specific summary sections to represent critical case information, thus expressing the case progress as transparently as possible (Michel, 2020, p. 39).

R3.5 Clarify Needed Contribution

Requirement: *For every case, professionals need to complete their assigned activities. Since professionals are traditionally responsible for multiple cases, the system shall provide a dashboard to indicate which case needs contributions from the professionals.*

Motivation: The collaborative and knowledge-intensive characteristics of case works imply the need for documenting various individual decisions made by caseworkers. For all caseworkers, their needed contribution and responsibility must be transparent to ensure the completion of case works (Herrmann & Kurz, 2011; Kurz, 2013, pp. 89-90; Hauder et al., 2014b, p. 26). Furthermore, the system shall allow the update of caseworker responsibilities to represent their actual duties in the case execution (Michel, 2020, p. 39).

4.3 Architecture

To articulate the design of SACM meta-model-based architecture, Section 4.3.1 describes the hierarchical layers and their functionality in the SACM backend. Furthermore, Section 4.3.3 illustrates the integration of SACM into CONNECARE, a European-funded integrated care application for chronic diseases used by four medical institutions in the Netherlands, Spain, and Israel. The CONNECARE integrated environment demonstrates how SACM collaborates with a central authentication authority and another subsystem to deliver a health monitoring service.

4.3.1 Conceptual Layers of SACM Backend

Figure 4.3 illustrates the latest version of the eight-layer architecture of SACM (Hernandez-Mendez et al., 2018, pp. 263-266). In summary, a single relational database stores all persisted data. These data enable linking elements of different conceptual layers to create a *domain ontology*. The backend exposes publicly accessible functionality via a RESTful API which authorizes requests based on roles and access rights. In this hierarchy, the higher conceptual layer provides higher abstracted functionality by leveraging the capabilities of the lower layer. This section successively describes each layer in detail regarding their roles in ACM. Each layer has a designated color to demonstrate its association with a given functionality or meta-model element (See Section 4.4). The description begins from the lowest to the highest layer.

■ **Annotated Versioned Linked Content Graph:** According to Hernandez-Mendez et al. (2018, p. 264), this layer supports versioned data storage and ensures the system can store revisions of semi-structured data as *entities* (e.g., JSON Objects) and *references* between them. The rationale is that organizations already apply structured or unstructured data in their operation; thus,

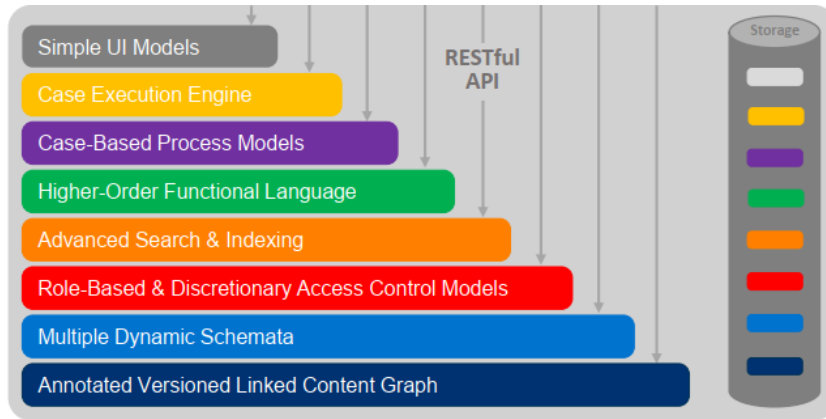


Figure 4.3: Conceptual architectural layers of SACM (Hernandez-Mendez et al., 2018, p. 263)

the system needs to import an *evolvable, schemaless* set of semi-structured data initially (e.g., Excel sheets, Wiki pages with embedded tables and media). This layer supports the data-first (schema-second) data modeling approach (Büchner, 2007).

■ **Multiple Dynamic Schemata:** Thanks to this layer, users can collaboratively structure their information over time by adding or removing schema constraints containing relationship and cardinality restrictions (Hernandez-Mendez et al., 2018, p. 264). Therefore, a data schema is a set of constraints applied to a flexible and evolvable content graph; rather than a container to store data subsequently (like a SQL Database). However, this implies that data can contain inconsistencies during the system evolution, which can be resolved collaboratively without leaving the system scope (Büchner, 2007; Matthes et al., 2011).

■ **Role-Based and Discretionary Access Control Models:** This layer addresses the security concerns in enforcing access rights in the system. The security model comprises internal or external users and groups of an organization. The access rights (administrator, editor, author, and reader roles) are defined initially at the *Workspace* level. They can be overwritten (loosened or tightened) at the *Entity* level to guarantee secure access to the data stored in the system. (Hernandez-Mendez et al., 2018, p. 264)

■ **Advanced Search and Indexing:** On this layer, the unstructured data (i.e., long texts or hypertexts) is linked to the semi-structured data (using parsing and full-text indexing technologies). This mechanism is a core element of the Hybrid Wiki approach (Matthes et al., 2011) and lays the groundwork for natural language processing techniques and model discovery processes. The rationale behind this design is that not all the data in the enterprise are structured or semi-structured, and the system should be able to use all possible data formats without the need for an upfront structuring process. This layer's description originates from Hernandez-Mendez et al. (2018, p. 264).

■ **Higher-Order Functional Language:** According to Hernandez-Mendez et al. (2018, p. 265), this layer introduces a strongly-typed query language (similar to LINQ - (Meijer et al., 2006)) to access all the structured information in the information system. The language allows 1) adding computed results as attributes of Entities (i.e., Derived Attributes) and 2) executing operations (e.g., Map, Reduce, and Join) over collections of Entities. The rationale is to offer the users a flexible mechanism to extract knowledge from the stored data, which is not limited by the basic API of the system, and allows the users to create adaptive and tailored data views based on individual information requests (Reschenhofer & Matthes, 2016, p. 98).

■ **Case-Based Process Models:** Hernandez-Mendez et al. (2018, pp. 265-266) states that this layer enables the user to define knowledge- and data-intensive processes following the adaptive case management paradigm introduced by Swenson and Palmer (2010). SACM uses and adapts the CMMN 1.1 (Object Management Group, 2016) specification as a reference to create the conceptual design (Michel, 2020, p. 43). According to Michel (2020), a modeler can define Stages, Tasks, and Sentries to declare a collaborative process. Each task links to one or multiple read-only or writable attributes. The design enables modelers to define and dynamically instantiate standard processes or reusable process fragments when needed. Modelers declare case templates with required data structures as XML files. Afterward, modelers import these files into the system for execution.

■ **Case Execution Engine:** This layer manages the information regarding 1) the states of all case instances being executed in the system, 2) the links to the shared or case-local data, and 3) the actors involved in a case (Hernandez-Mendez et al., 2018, p. 266). Moreover, the case execution engine enforces the access control policies defined in the access rights layer. Meanwhile, the data management layer keeps an audit trail of the executed steps and data modifications (Hernandez-Mendez et al., 2018, p. 266). The rationale is to monitor the executed steps in the process and how the users accomplished the case. These data are helpful for future analyses, such as compliance auditing or prediction. In SACM, this layer provides communication and coordination features, such as case-based messaging, notes, and alerts (Michel, 2020, p. 43).

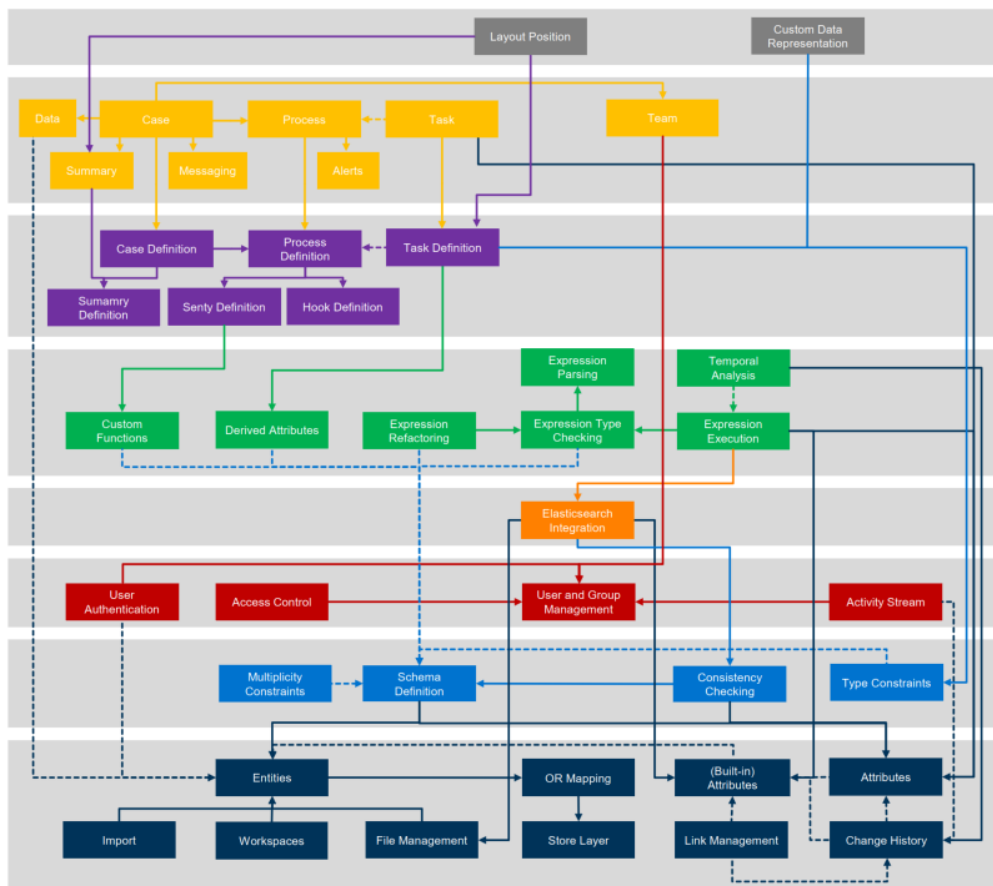


Figure 4.4: Capabilities ordered by conceptual layers. Solid lines represent the usage of a specific capability. Dashed lines represent extended functionality. Adapted from Hernandez-Mendez et al. (2018, p. 265) by Michel (2020).

■ **Simple User Interface Models:** According to Michel (2020, p. 43), this layer enables custom data representation on the user interface. The schemata layer provides basic data types, with each data type being bound to strictly one generic readable and writable representation. In addition, this layer supports overriding the default representation and grid layout option to provide a custom data representation.

Figure 4.4 illustrates the capabilities of each layer and their usage or extension relationships (Hernandez-Mendez et al., 2018, p. 265). The color of an element or capability represents its association with a layer in Figure 4.3. Section 4.4 further describes the hierarchy and features of the conceptual meta-model.

4.3.2 The CONNECARE Project

The decision support and case management features of SACM power the operation of the Personalized Connected Care for Complex Chronic Patients research project (CONNECARE). The European Union's Horizon 2020 research and innovation program funded 5 million Euros for CONNECARE under grant agreement No. 698802 from April 2015 to December 2019. The project seeks to provide integrated care solutions for patients with chronic diseases.

Objective: CONNECARE aims to improve care coordination and self-management of Complex Chronic Patients (CCPs) by co-designing, developing, deploying, and evaluating a novel integrated care services model supported by a smart and adaptive case management system (Vargiu et al., 2017). A CCP is "a patient with at least one chronic diseases, comorbidities, frail (due to social, economic and/or clinical factors), usually elderly, and who consumes a very high level of health resources" (Vargiu et al., 2017). An estimation of over 40 percent of all hospital admissions is required to provide the healthcare needs for CCPs.

Consortium: The CONNECARE comprises four clinical and five technical partners. The following clinical partners executed implementation studies within their hospitals: 1) University Medical Center Groningen (UMCG), located in Groningen, Netherlands. 2) Assuta Medical Centers (ASSUTA), located in Tel Aviv, Israel. 3) Institute de Recerca Biomèdica de Lleida Fundació Dr Pifarré (IRBLLEIDA) located in Lleida, Spain. 4) The Consorci Institut D'Investigacions Biomediques August Pi i Sunyer (IDIBAPS), located in Barcelona, Spain.

Meanwhile, the following technical partners developed or provided information technology solutions to perform the implementation studies: 1) Eurecat Technology Center (EURECATE), located in Barcelona, Spain. 2) Technical University of Munich (TUM), located in Munich, Germany. 3) Advanced Digital Innovation UK Ltd (ADI), located in West Yorkshire, United Kingdom. 4) Università degli Studi di Modena e Reggio Emilia (UNIMORE), located in Modena, Italy. 5) eWave, located in Tel Aviv, Israel.

Stakeholders: Figure 4.5 demonstrates the CONNECARE vision to orchestrate patients, informal carers (e.g., patient relatives), and responsible professionals across organizational boundaries who conducted the *patient-centric treatment* (Michel, 2020, p. 135). Clinical partners of the projects further classify professionals into *hospital staff*, *specialist doctors*, *primary care doctors*, and *social workers*. The system must coordinate *patient-centric treatment* among all professionals involved and communicate the treatment to the patient.

4 Smart Adaptive Case Management (SACM)

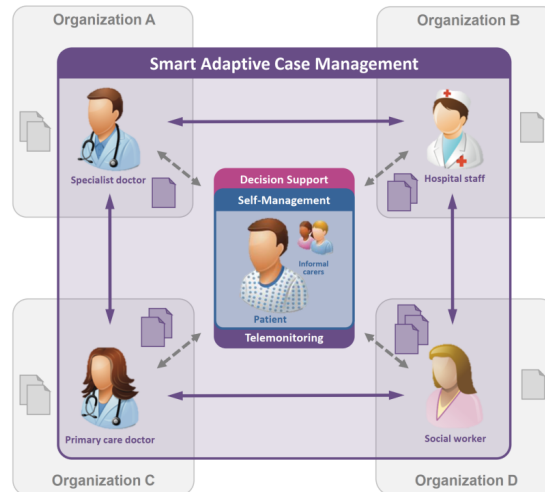


Figure 4.5: High-level project vision. The Smart Adaptive Case Management provides care services with its collaborative, purely meta-model-based approach enriched with clinical decision support (Michel, 2020, p.135).

Patient-centric Treatment Operation: The regional execution of the collaborative management of CCPs follows a protocol described by Cano et al. (2017). Implementing the protocol requires supporting professionals with different integrated care concepts, such as *telemonitoring*, *self-management*, and *decision support* (Michel, 2020, p. 135).

Specifically, *telemonitoring* assists in the inceptive detection of anomalies, enabling pre-emptive, immediate care activities. The *Self-Management System* (SMS) empowers patients to participate in care activities actively; record their medical, mental, or physical conditions through questionnaires and tasks execution; and communicate with the responsible care professionals (Vargiu et al., 2019). Therefore, the SMS is crucial for medical institutions to monitor health conditions and understand the concerns and satisfaction of the patients (Vargiu et al., 2018, 2019).

Meanwhile, the *decision support system* 1) assists professionals in risk assessment and stratification to focus on acute cases, 2) visualizes patients on a map based on their health conditions, barriers to treatment (e.g., anxiety, low income), optimal route for a home visit, and relevant medical facilities, and 3) design CP using customizable templates (Mariani et al., 2019). The system also provides a recommender to communicate the patient’s adherence to treatment and engage patients to play an active role in their care process (Fernández et al., 2017, p. 2; Mariani et al., 2019). As a result, the project vision forms a three-dimensional care paradigm that includes a medical organization, care services, and an underlying technological infrastructure (Michel, 2020, p. 135).

4.3.3 SACM-CONNEXARE Integration

In the CONNECARE project, SACM serves as a component that provides the case management and decision support features to medical professionals via a web-based UI. Figure 4.6 illustrates the CONNECARE architecture with user roles, components, and neighbor systems. The remainder of this section describes the purpose of each component in the system.

SACM: With an ACM4IC engine, SACM provides core backend functionality such as case management, external system communication, and collaboration features as described in the Re-

4 Smart Adaptive Case Management (SACM)

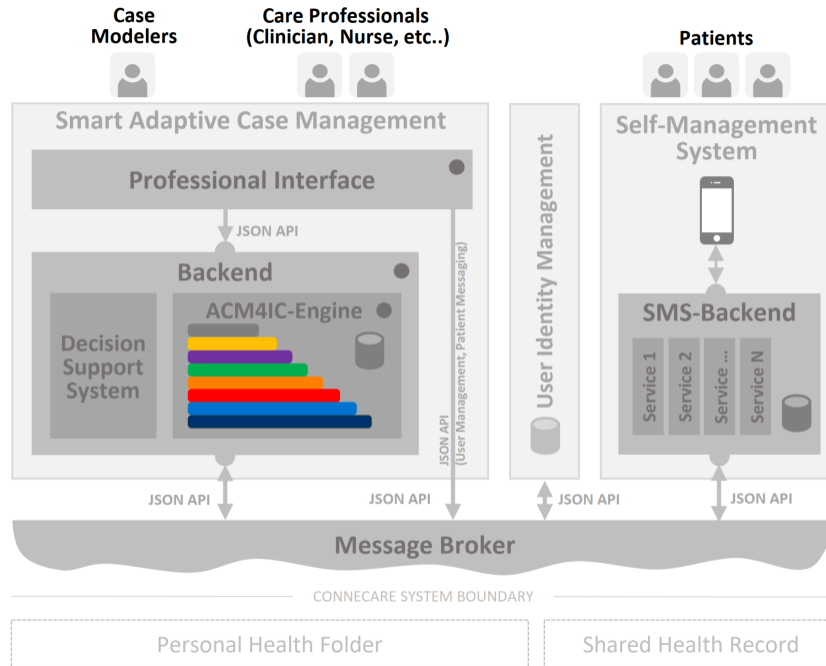


Figure 4.6: The CONNECARE system architecture adapted from (Michel & Matthes, 2018, p. 17). ACM4IC Components are highlighted with a dot in the upper right corner (Michel, 2020, p. 141)

quirement Section (See Section 4.2). ACM4IC is a Java application that supports modeling capabilities across different conceptual layers (e.g., case entities, access control, UI), as Figure 4.6 illustrates. The purely meta-model-based approach of SACM enables reusing individual components in different contexts, such as data representation in the frontend or integration with existing systems. Additionally, SACM contains a *decision support system* to provide supplemental risk information for a patient.

SACM exposes the backend functionality of those layers via a JSON-based API. Therefore, a Single Page Web Application powered by Angular version 5.2.10 accesses the backend API to provide the features for medical professionals.

SMS: The users of SMS are patients. The backend of SMS consists of microservices to provide self-management features, e.g., patient monitoring prescriptions. The patients use the features via an Android or iOS app developed with Xamarin to maintain a single code base for both mobile platforms.

SACM interacts with SMS microservices using the DualTask meta-model element. This integration method enables case models in SACM to connect to hardwired SMS microservices repeatedly and flexibly.

User Identity Management (UIM): A centralized interface provides the Single Sign-On (SSO) service for all users in the CONNECARE system. Moreover, the UIM is responsible for creating, updating, and disabling users in the CONNECARE system (Michel, 2020, p. 141). Regarding the user creation, the UIM adds the created user internally and pushes that user to the SACM. The system applies a similar process to other operations on the users.

4 Smart Adaptive Case Management (SACM)

Regarding authentication, the UIM generates a *JSON Web Token (JWT)* with a *private key* for each successful authentication. All other components validate the JWT using a *public key*. This JWT generation mechanism simplifies the integration of all services on all architectural levels (Michel, 2020, p. 140). For example, the professional UI of SACM can directly request the UIM API endpoint for authenticating a user.

Message Broker: A RabbitMQ *message broker* regulates communication between SACM and SMS by providing *producer* and *consumer* API endpoints. Additionally, the *message broker* receives necessary information from adapters of third-party site-specific hospital systems. This approach enables integration with external systems. Due to legal regulations, CONNECARE only performs read operations on the external HISs (Michel, 2020, p. 140).

Table 4.1 summarizes the responsibilities of components in CONNECARE (Michel, 2020, p. 142). A *Master* role indicates that the component primarily orchestrates the activity, while a *Slave* role means the component passively collaborates with the *Master* and may need to synchronize data with other components.

	UIM	SACM	SMS
Authentication	Master <i>creates JWT token</i>	Slave <i>validates JWT token</i>	Slave <i>validates JWT token</i>
User Creation User Update User Disable	Master <i>user basic fields</i>	Slave <i>synced users, pushed on change by UIM</i>	Slave
Case Authorization	N.A.	Master	Slave
Case Instantiation Case Update Case Delete	N.A.	Master <i>provides hooks for integration</i>	Slave <i>use hooks to sync</i>

Table 4.1: High-level responsibilities of components in the CONNECARE system (Michel, 2020, p. 142)

To illustrate the coordination of the SACM and SMS, Figure 4.7 shows the conceptual flow of a monitoring prescription scenario (Michel, 2020, p. 143). First, nurse Anne Williams adds a *DualTask* to a *Workplan* Stage. This *DualTask* requests the patient to measure their *body temperature once a day* within a period specified by a start and end date. The *DualTask* also contains a range of healthy *body temperatures* with a minimum and maximum value. After SACM validates and stores the *DualTask*'s input parameters, it checks if any hook execution is needed when the *case state* changes. Since SMS needs to display this monitoring task to the patient, the SACM *DualTask* has an *HTTP Hook definition* to transmit the task data *upon completion* to the SMS as a JSON serialized payload. The SMS then pushes the new monitoring prescription request to its mobile app.

Next, patient Maya Wilson measures her *body temperature* with a wearable thermometer. The SMS mobile app then forwards the recorded *temperature* to its backend, which drafts the monitoring prescription task in the SACM backend. The nurse can then read the *body temperature* in the SACM UI almost in real-time.

If the patient's *temperature* exceeds the defined minimum and maximum value, the SMS notifies SACM to alert the related task owners. As a result, nurse Anne Williams will see a new notification when she opens her dashboard. In case the nurse contacts the patient and discovers that the *temperature* is much lower than the current value, she executes the *task correct feature* in SACM

4 Smart Adaptive Case Management (SACM)

to edit the *temperature*. After that, the SACM backend receives the corrected information and, if specified, can update this value in the SMS by sending a hook request.

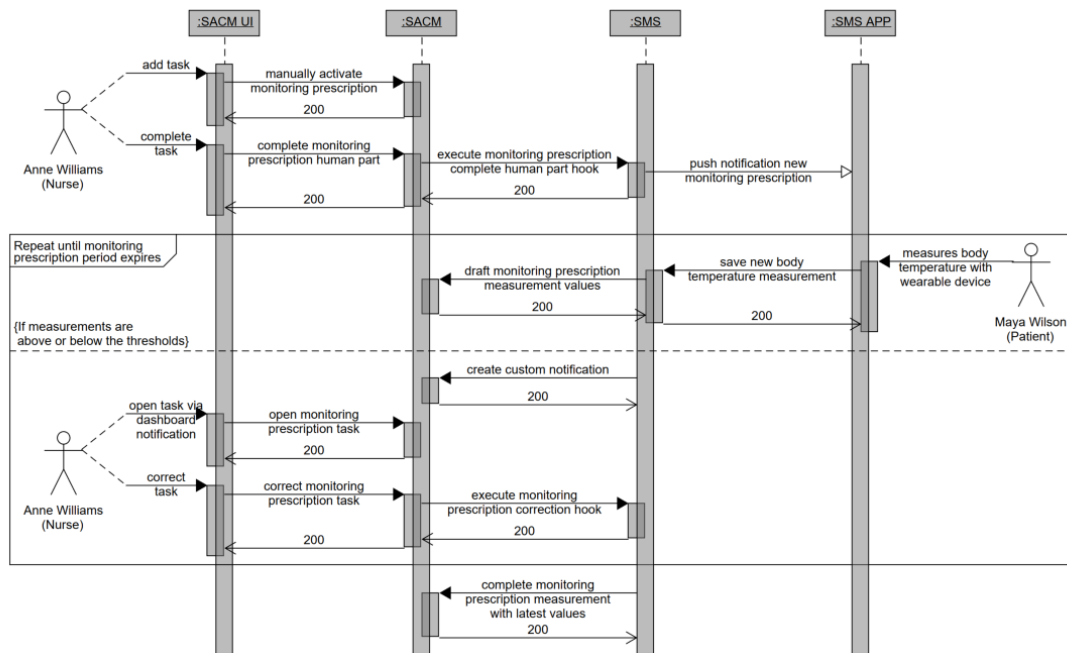


Figure 4.7: Conceptual orchestration of a monitoring prescription task (Michel, 2020, p. 143).

4.4 Meta-model Elements in SACM

The conceptual meta-model in Figure 4.8 illustrates the modeling concepts in each conceptual architecture layer (See Figure 4.3) and their dependencies with other layers. The UML class diagram meta-model focuses on relevant concepts for designing a holistic ACM and simplifies other related notions. In the diagram, a gray box represents the boundary of an *architecture layer*. Meanwhile, the colored modeling elements indicate the specific layer they belong to according to the color code described in Section 4.3.1. The following paragraphs describe the modeling elements in the context of their architecture layer and ACM (Michel, 2020, pp. 44-48).

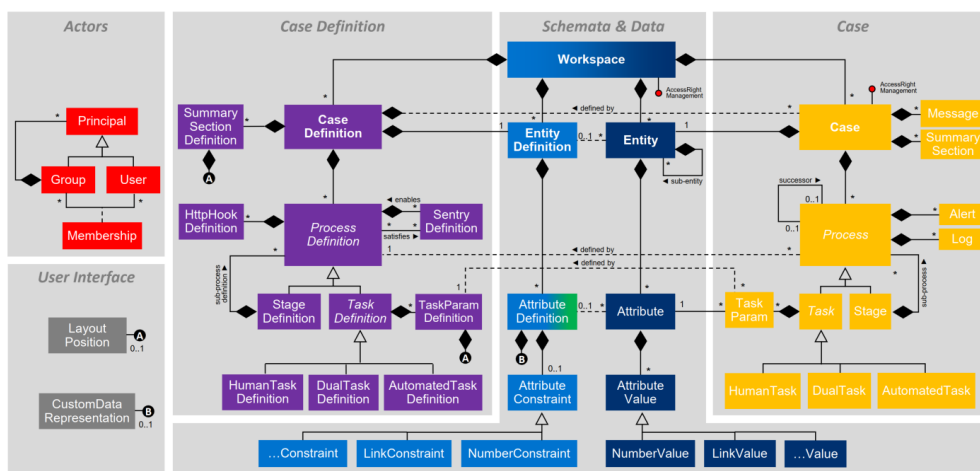


Figure 4.8: Meta-model adapted from Hernandez-Mendez et al. (2018, p. 267). A complete conceptual meta-model containing relevant attributes is illustrated in Figures A.6, A.7, and A.8 of Appendix A.7

4.4.1 ■ Schemata and ■ Data

A Workspace is a container storing definitions of model elements and their instances. In addition, the Workspace might declare access rights, including *readers*, *writers*, *administrators*, and *contributors*. Among them, the *contributor* role might have the right to *instantiate* and get direct *write access* to model elements. Typically, a Workspace defines a protected environment for a specific stakeholder Group to access. For example, in integrated care, stakeholders are care professional teams, social workers, informal carers, patients, and administrators that can access their treatment cases.

The meta-model leverages the combination of EntityDefinition and AttributeDefinitions to resemble a data structure of a class and its attributes in the UML notation (Object Management Group, 2015, p. 192). Likewise, an Entity and its Attributes represent an object and its attribute value. SACM supports late modeling using a *data-first and schema-second* strategy thanks to the loosely coupled data and schemata layers. However, the system does not combine late modeling with case management.

SACM associates one AttributeDefinition with only one EntityDefinition (Michel, 2020, p. 45). Likewise, one Attribute is associated with precisely one Entity of the Case. The meta-model supports multiple AttributeValues based on the specified *multiplicity* having the following values: *any*, *maximum one*, *exactly one*, or *at least one*. Each AttributeDefinition may have an optional AttributeConstraint to define its type, therefore enforcing which AttributeValue is valid at the instance level. The supported primitive types for AttributeValues are *string*, *longtext*, *number*, *date*, *enumeration* (multiple choice), *JSON object*, and *references* to an object (e.g., Entity, Group).

An AttributeConstraint subsumes several constraints. First, the NumberConstraint only accepts a NumberValue with optional minimum and maximum values. The EnumerationConstraint enables declaring multiple choice questions with multiple options. Each option contains a human-readable *description* and a *value* used for internal computation. For example, an option "Yes" has a value of "1", and an option "No" has a value of "0". Finally, the LinkConstraint restricts a LinkValue, which is a reference to a valid instantiated model element, such as an Entity, Principal, Group, or User. A LinkConstraint referring to an Entity can optionally declare the accepted EntityDefinition. In addition, a LinkConstraint referring to a User can optionally declare a set of Groups in which the User must have a Membership in at least one of the Groups.

Besides AttributeDefinition, the meta-model provides DerivedAttributeDefinitions to declare dynamic calculations and expressions of output based on input values, typically from AttributeDefinitions. To simplify the visualization, the DerivedAttributeDefinitions is a part of the AttributeDefinition painted with a green background. Conceptually, DerivedAttributes are not referable with a unique identifier because SACM evaluates all DerivedAttributes dynamically.

4.4.2 ■ Actors

All access rights in a Workspace accept one or multiple Principals as value. The role structure follows a composite pattern: the Group is a composite, the User is a leaf, and the Principal is an abstract component. A Membership specifies that a User is a member of a Group. Additionally, Membership can include meta-data such as the user creation and expiry dates. This hierarchy enables nesting Groups in the form of Group Membership.

By default, the containing model elements of a Workspace inherit its access rights. However, modelers can define exceptional access policies. Furthermore, certain Cases can extend the inherited access right with additional access rules.

4.4.3 ■ Case Definition

A CaseDefinition is a container of all model elements that collectively constitute a holistic, purely meta-model-based Case template (Michel, 2020, p. 46). Each CaseDefinition refers to only one EntityDefinition that represents the root data structure of the Case. Optionally, the Case root Entity is extensible by attaching a second EntityDefinition, which declares the 1) roles accessible to the Case and 2) due dates applied to workflow elements, such as Stages or Tasks. This second EntityDefinition can serve as the Case Settings. In addition, each CaseDefinition has an *owner path* which refers to the Group or User responsible for managing the Case. Likewise, a Case declares a *client path* referring to the treated patient.

A CaseDefinition has a *version* attached to it. When importing the CaseDefinition, SACM automatically sets the *instantiable flag* of older versions to *false* by default. Furthermore, an optional *notes template* is declarable inside the CaseDefinition.

The ProcessDefinition represents the abstract of StageDefinition and StageDefinition in the composite pattern. ProcessDefinition contains essential properties for declaring a Process in the workflow. By default, a Process executes only once, yet they can be *serially* or *parallelly repeatable* when defined. Furthermore, a Process has a flag to indicate whether it is *mandatory* to complete the Case or parent Process. SACM automatically activates a Process concerning its constraint, i.e., SentryDefinitions. Otherwise, the system supports declaring *manual* or *expression-based activation*, which evaluates a manually or automatically activated Process at runtime based on the expressed conditions.

Additionally, each ProcessDefinition contains an *owner path* which refers to a User or Group responsible for the Process accomplishment and handling of unpredictable events. This *owner path* depends on the Case root Entity and is resolved during runtime. Finally, since Processes commonly generate and extend the existing Case data, each Process contains a path referring to its attached EntityDefinition, which extends the Case Entity structure.

The composite pattern of the workflow structure supports the declaration of nested workflow elements. The ProcessDefinition represents the abstract of StageDefinition and TaskDefinition in the composite pattern. The StageDefinition is the composite that contains TaskDefinitions as the leaf. A TaskDefinitions is either 1) an AutomatedTaskDefinition representing AutomatedTask executed by a third-party system, or 2) a TimedTaskDefinition representing HumanTaskDefinition or DualTaskDefinition. A time constraint is applicable to both task types by declaring a *due date path*. The executors of a HumanTask are human caseworkers, while a DualTask is a combination of HumanTask followed by an AutomatedTask.

A TaskDefinition contains TaskParamDefinitions to declare how SACM visualizes the inputs or outputs of the Task Parameters in the UI. In addition, each TaskParamDefinition has 1) a *read-only* property to define if the parameter is writable or only readable, 2) if the parameter is *mandatory* for the Task completion, and 3) the declaration of a path that links to the Attribute of a Task Entity, starting from the Case root Entity. This method forms a Task data structure by binding Attributes to TaskParams. This binding enables updating previous results of the Task.

Modeling complex preconditions for ProcessDefinitions (i.e., StageDefinitions and TaskDefinitions) is possible thanks to SentryDefinitions (Michel, 2020, p. 47). Specifically, a SentryDefinition pre-

4 Smart Adaptive Case Management (SACM)

condition declares which ProcessDefinition(s) must be completed before activating another Process. Optionally, a conditional expression on elements of the data layer enables defining control flow among Processes. For example, a Case only triggers a *Prescription* Stage when the blood pressure value of the *Evaluation* Stage is *not Normal*. SACM only considers these expressions after the ProcessDefinition preconditions are satisfied.

SACM leverages HttpHookDefinitions to orchestrate integrated processes with systems from CONNECARE or third parties. A HttpHook is an HTTP Request containing the URL of the destined system; an HTTP method value of *POST*, *GET*, *PUT*, or *DELETE*; optionally, a *failure message* is declarable. The HttpHookDefinition defines a HttpHook to send an HTTP request when a specified Process state change event occurs, e.g., *activate*, *enable*, *complete*, *terminate*, or *Task correction* event. For example, SACM only sends an HTTP POST request to add the newly *measured blood pressure* values to a partner system when the *blood measurement* Task is completed. A HttpHook in CaseDefinition is more restricted because one Case event declares only one HttpHook, which only supports the GET method and can not define the *failure message*.

For the provision of a customizable Case data summary, the meta-model offers a SummarySectionDefinition declaration, each containing a *name* and a *path* to an Attribute on the *data* layer. As a result, a SummarySectionDefinition displays the Attribute's value of a specific Case data, e.g., medical status or patient's personal information. In addition, SACM supports a customizable grid layout definition to arrange SummarySectionDefinitions, as described in the following subsection.

4.4.4 ■ User Interface

The meta-model enables modelers to specify the LayoutPosition of Summary Sections and Task Parameters in a three-column grid layout. The supported positions are *left*, *center*, *right*, and *stretched* (i.e., cover the entire width of the grid layout area). Additionally, TaskParamDefinitions have more complex layout positions, such as *left-center* and *center-right*, which stretch over two columns. By default, the purely meta-model-based approach binds each AttributeConstraint to exclusively one representation in the user interface (Michel, 2020, p. 47).

To enable customization of data representation that complies with the meta-model-based approach, SACM provides CustomDataRepresentations to display the data as an SVG graphic by specifying a string; or a line diagram containing data nodes in the JSON format. This ability enables modelers to extend the CustomDataRepresentations based on domain-specific needs.

4.4.5 ■ Case

The Case execution engine instantiates most modeling elements stated in the Case Definition layer (Michel, 2020, pp. 47-48). Specifically, one Case refers to 1) its root Entity; 2) the *case owner* and *client* paths, each path pointing to a User or Group; 3) Principals containing the dedicated Case *readers* and *writers*; 4) the Messages created during the Case execution. Thus, a Message exists only in the Case instance and contains the creation date and a User as its author; 5) Summary Sections to recapitulate critical case data; and 6) all Process elements, i.e., Stages and Tasks.

Regarding the Process, each has a property to indicate the current state of its lifecycle, which contains the following states (Michel, 2020, p. 49):

4 Smart Adaptive Case Management (SACM)

- **Available:** The Process is instantiated but is not enabled because it does not meet a *precondition*. The reason can be 1) inactive parent Stage, 2) The Process's SentryDefinition is not satisfied, 3) a combination of 1) and 2).
- **Enabled:** The Process meets its *precondition*, i.e., having an active parent Stage and its SentryDefinition is satisfied. However, the Process requires a manual activation to execute runtime planning.
- **Active:** The Process is currently executed, and its data are still modifiable. In other words, the User may input values to several TaskParams, but they did not complete nor terminate the Task.
- **Terminated:** The Process is not finalized successfully because 1) the caseworker deems the Process not suitable for the Case, hence terminating it; 2) an external system can abort a Process when it is no longer applicable.
- **Completed:** The Process is finalized successfully. This state occurs when 1) a caseworker executed all the required actions and completed the Process; 2) an external system completed an AutomatedTask.

When processing a repeated process, SACM instantiates multiple Processes and associates the successor Process with its predecessor to form a repetition order. The Process inheritance structure on the case execution layer resembles the one in the case definition layer. Specifically, a TaskParam maps to the TaskParamDefinition with a path linking to an Attribute. Additionally, the abstract TimedTask has a path referring to a due date Attribute, which allows overriding the AttributeValue locally. Finally, a DualTask has two internal states to keep track of the current lifecycle state from the human part and the automated part.

Additionally, each Process contains Logs created at runtime to assist in debugging complex scenarios. The properties of a Log are 1) a creation date, 2) a log level, 3) a log message, and 4) an optional description property. Logs help debug the HttpHook execution of integrated Processes with third-party systems.

Another concept is Alert, which has three types in SACM (Michel, 2020, p. 48). First, an error Alert occurs when an HttpHook execution fails, thereby triggering the error Alert to show the defined *failure message* of the HttpHook. The second Alert type appears when a caseworker modifies a completed or terminated Task. Finally, an external system can trigger a custom Alert at any time. This Alert enables incorporating complex domain-specific logic into the microservices of partner systems.

Several modeling elements with composite associations have strongly coupled dependencies, thus enabling cascading delete operations. Therefore, deleting CaseDefinition results in eliminating all associated Cases.

5 Language Design

Acadela derives the features of modeling CPs based on the SACM metamodel and requirements from SACM and existing DSLs. As a result, Acadela incorporates both the essential elements of CP construction identified in the literature while inheriting SACM capabilities of external communication and customizable data presentation. The first section of this chapter presents an overview of the CP modeling capabilities of different DSLs in the Related Work chapter compared to Acadela. Afterward, the second section describes the concrete syntax to demonstrate how Acadela declares CPs in SACM and simplifies the current Case definition syntax.

Besides CP modeling, Acadela provides error validation, custom IDE, and CP visualization to support the modeling process. The remaining sections of this chapter discuss the handling of syntax and semantic errors in Acadela, the architecture to incorporate the CP modeling and supporting features, and a reflection on the rationales of the language design decisions.

5.1 Requirements for Modeling Clinical Pathways

Table 5.1 summarizes the CP modeling features from the DSLs discussed in the Related Work Chapter (See Chapter 3) and Acadela. The features are identified from the requirements of SACM and other DSLs for CP modeling. Since fostering communications among care professionals are not CP modeling functions, Table 5.1 excludes sub-requirements in SACM R3 (Section 4.2.3).

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	SF1	SF2
DSML4CPs	●	●	●	●	●	●	●	●	●	●	○	○	○	●	●
BPMN4CP	●	●	●	●	○	●	○	●	●	●	○	○	○	●	●
BPMN ^{SIX}	●	●	●	●	○	●	●	●	●	○	○	○	○	●	●
FCIG	●	●	◐	○	○	○	○	◐	●	○	○	○	○	◐	○
Acadela	●	●	●	●	●	●	●	●	●	◐	●	●	●	◐	●

○ No Support ◐ Partial Support ● Full Support

Table 5.1: Features Comparison of DSLs for CP modeling.

F1: The DSL shall model medical processes using concepts from the healthcare domains. These concepts shall be familiar to healthcare professionals.

Rationale: First, concepts used in the DSL should map to the terminology that the prospective users are familiar with (Frank, 2013, p. 136). Second, constructing *activities* and *phases* of a medical process is essential for care professionals to conduct their treatment.

Requirement Source: SACM **R1.2** Requirement (See Section 4.2.1), DSML4CPs **R1** (Heß et al., 2015, p. 5), BPMN4CP **R1** (Braun et al., 2014, p. 12), BPMN^{SIX} (Neumann et al., 2016, p. 3).

SACM Solution: Defining treatment Stages and Tasks using StageDefinition and TaskDefinition. The schema of each Stage or Task is an Entity with Attributes defined by EntityDefinition and AttributeDefinition. Using familiar medical terminology, modelers and care professionals can declare medical concepts in each Stage and Task and model them in the Entity schema.

F2: The DSL shall model alternative treatment processes, their triggering conditions, and potential values that could activate other alternatives (Heß, 2013, p. 375).

Rationale: Medical professionals frequently handle unpredictable scenarios during the planned treatment process. Therefore, a DSL shall provide decision support to healthcare professionals, patients, and their relatives to decide the appropriate medical intervention (De Bleser et al., 2006, p. 562).

Requirement Source: SACM **R1.2** Requirement (See Section 4.2.1), DSML4CPs **R2** (Heß et al., 2015, pp. 5-6), BPMN4CP **R6** (Braun et al., 2014, p. 12), BPMN^{SIX} (Neumann et al., 2016, p. 4), FCIG Computer-based Clinical Practice Guidelines (CPG) (Msosa, 2018, pp. 11-12).

SACM Solution: Defining a SentryDefinition for a Stage or Task results in an alternative treatment process. The alternative process is triggered only when it satisfies all conditions in the SentryDefinition.

F3: The DSL shall model evidence classes with data assigned to them (Heß et al., 2015, p. 6). Additionally, the DSL shall support referencing the evidence (Braun et al., 2014, p. 4).

Rationale: Medical experts leverage CPs to treat patients according to the principle of Evidence-Based Medicine (EBM) (S. Wang et al., 2011; X. Wang et al., 2021). Therefore, CP processes and decisions shall be supplemented with information on the evidence to decide the appropriate interventions.

Requirement Source: DSML4CPs **R3** (Heß et al., 2015, p. 6), BPMN4CP **R3** (Braun et al., 2014, p. 12), FCIG (Msosa, 2018, pp. 7-8).

SACM Solution: As described in **F1**, the data layer of SACM serves as a schema to store information of particular evidence collected in a specific Task.

Remark: FCIG can process references to a medical evidence, yet the DSL does not support declaration of an evidence class.

F4: The DSL shall model iterative treatment processes. Parallel repetition should also be supported (Michel, 2020, p. 46; Neumann et al., 2016, p. 4).

Rationale: CP shall support control flow by executing both optimal, sequential procedures and repetitive, unpredictable processes in a treatment. Parallel execution is necessary to construct simultaneous tests of the same type (Michel, 2020, p. 46) (e.g. parallel PCR tests (Berdal et al., 2008; Perchetti et al., 2020, p. 2).

Requirement Source: SACM **R1.2** and ProcessDefinition (Michel, 2020, pp. 34, 46), DSML4CPs **R4** (Heß et al., 2015, pp. 6), BPMN4CP **R2** and **R5** (Braun et al., 2014, p. 12), BPMN^{SIX} ParallelFlow (Neumann et al., 2016, p. 4).

SACM Solution: The ProcessDefinition has an *repeatable* attribute to specify if a Stage or Task is sequentially executed or repeated serially/parallelly.

F5: The DSL shall model each medical process's treatment goal(s) based on medical practices and preferences of individual patients.

Rationale: Expressing goals based on the applied remedies and the patient's preference fosters treatment transparency (Heß et al., 2015, p. 6). Furthermore, this feature encourages medical professional teams to establish unified treatment approaches to achieve the care goal(s) (Heß et al., 2015, p. 6).

Requirement Source: DSML4CPs **R5** (Heß, 2013, p. 6).

SACM Solution: the DerivedAttributeDefinition can express the condition for accomplishing a Task's goal based on the TaskParams, which store the patient's preferences or medical conditions. Furthermore, SACM can configure the goal's text and UI decoration to display it on the GUI of a Task or SummarySection.

F6: The DSL shall support assigning responsibilities to medical processes (at runtime). The responsibilities distinguish between staff that executes medical tasks and staff responsible for the treatment outcome.

Rationale: All caseworkers (care professionals) shall know their required contributions to the treatment. Therefore, responsibility must be transparent to ensure the completion of case works (Herrmann & Kurz, 2011; Kurz, 2013, pp. 89-90; Hauder et al., 2014b, p. 26). Specifying the responsibilities of each healthcare professional optimizes the sequencing of medical procedures, indicates contact persons, and fosters communication among staff (Heß et al., 2015, p. 6). Furthermore, since treatment activities are unpredictable, responsibilities cannot be assigned upfront (Michel, 2020, p. 35). Thus, a DSL should support the declaration of responsibilities for case activities at runtime.

Requirement Source: DSML4CPs **R6** (Heß et al., 2015, p. 6), SACM **R1.3** (Michel, 2020, p. 35), BPMN4CP **R1** (Braun et al., 2014, p. 12), BPMN^{SIX} (Neumann et al., 2016, p. 6).

SACM Solution: The Principal in SACM declares the participating Groups or Users representing *departments, care teams, or medical professionals*. The Case access rights of each Principal are *read, write, and owner* to reflect different levels of intervention or management. In addition, each CaseDefinition, StageDefinition, and TaskDefinition includes an *ownerPath* to define the Groups or Users having the right to execute the Case, Stage, or Task.

F7: The DSL should model checklists associated with medical processes.

Rationale: Checklists are widely accepted and helpful instruments to show details of medical process executions (Healey et al., 2011, p. 3; Wolff et al., 2004, pp. 430-431).

Requirement Source: DSML4CPs **R7** (Heß et al., 2015, p. 6).

SACM Solution: An EntityDefinition of enumeration type can define a checklist as a *multiple-choice* or *single-choice* questionnaire (Michel, 2020, p. 80) by specifying the *multiplicity*

property to *any* or *exactlyOne*, respectively. SACM EnumerationConstraints define the label for each *choice* and their associated numeric or text values (See Section 4.4.1).

F8: The DSL should model various aspects and information needs of stakeholders in medical institutions. Therefore, the DSL should support inclusion of medical documents (guidelines, forms, etc.) necessary to accomplish medical processes.

Rationale: CPs aim to optimize medical care from various perspectives. Therefore, Acadela should provide concepts to model treatment-related information, such as (multidisciplinary) medical or organizational aspects (Heß et al., 2015, p. 6). Furthermore, the DSL should include documents necessary for the treatment to assist care professionals.

Requirement Source: DSML4CPs **R8** (Heß et al., 2015, p. 6), BPMN4CP Document Concepts (Braun et al., 2016, p. 3252), BPMN^{SIX} (Neumann et al., 2016, p. 4).

SACM Solution: The combination of EntityDefinitions and AttributeDefinitions enables the definition of generic objects; thus, it can model different information needs, such as patient consent or patient data collected during admission or discharge Stages. Furthermore, SACM also supports declaring a TaskParam containing an URL path to a document, thus CPs can include various document types (e.g., consent forms, medical guidelines) into the process.

F9: The DSL should model the allocation of resources to the medical process.

Rationale: CP's goals include supporting the management and optimizing resource consumption (Heß et al., 2015, p. 6). Resources comprise equipment or documents used during the medical process (Neumann et al., 2016, p. 4), medicine, and facilities (Heß et al., 2015, p. 6; Braun et al., 2016, p. 3252).

Requirement Source: DSML4CPs **R9**, BPMN4CP (Heß et al., 2015, p. 6; Braun et al., 2016, pp. 3251-3252), BPMN^{SIX} (Neumann et al., 2016, pp. 3-4).

SACM Solution: SACM can model medical resources such as drugs, aids as an Entity, with the resource properties as Attributes. Moreover, SACM can request a server to update the consumption of medical resource using `HttpHook`, which sends the necessary information of the resource and quantity to a server API that checks for the resource availability and updates the in-stock value based on the consumed units.

F10: The DSL should model time constraints and explicit time events.

Rationale: the definition of CP incorporates the timing necessary to achieve care goals (De Bleser et al., 2006; Campbell et al., 1998; Vanhaecht et al., 2006). Therefore, a DSL should provide features to declare due dates or timed activities such as periodic repeated or time-lapsed tasks.

Requirement Source: DSML4CPs **R8**, (Braun et al., 2014, p. 12), BPMN4CP **R4** (Braun et al., 2014, p. 12).

SACM Solution: Defining a *due date* as an AttributeDefinition in the Case Setting's Entity-Definition sets a variable for default time constraint (e.g., 24 hours, one week). Afterward, a TimedTaskDefinition (HumanTaskDefinition or DualTaskDefinition) includes the path to this *due date* variable, enforcing the time constraint when activating the Task. However, SACM does not support time-lapsed events and periodically repeated Tasks within a Stage. Nev-

ertheless, SACM can declare a repeatable Stage containing tasks with a *due date* to model periodic and repetitive activities.

F11: The DSL should model external requests to orchestrate and integrate with existing or external systems.

Rationale: e-Health services may need to interact with legacy systems which store user data. Furthermore, delivering care services may require sharing medical data with partner systems at different lifecycles of the activities. Therefore, a DSL should support defining external request communications when a process reaches a specific state.

Requirement Source: SACM R2.2, R2.3 (Michel, 2020, p. 37).

SACM Solution: Defining a `HttpHookDefinition` inside a `StageDefinition` or `TaskDefinition` triggers a HTTP request to an external system or service at a given lifecycle state. This request sends all information of the Stage or Task to a specified URL when reaching a defined state at runtime. An HTTP method and a custom error message are also declarable in the `HttpHookDefinition`.

F12: The DSL should support customizing the representation of an activity.

Rationale: Customization of the UI display of activities enhances extendability and supports the modelling of specialized clinical use cases (Michel & Matthes, 2018). Additionally, care professionals need a holistic, insightful view of the patient's medical status on the user interface to ease the analysis the patient's condition and planning an appropriate intervention (Suganthi & Poongodi, 2021).

Requirement Source: SACM R1.4 (Michel, 2020, p. 36).

SACM Solution: SACM provides five UI customization features: 1) specifying the layout position of a Task element (`TaskParamDefinition`), 2) defining color background applied to different numeric ranges of a `TaskParam` value, 3) conditionally outputting a value based on other `TaskParams` of the same Task, 4) applying SVG with adaptive styling based on other `TaskParam(s)` value, 5) displaying a collection of data with time series as a line chart.

F13: The DSL should support importing CP elements into the current CP definition.

Rationale: Different CPs may use the same administrative or medical tasks, e.g., patient admission, laboratory test. Repeatedly declaring these tasks in each CP results in duplication and cumbersome modification for all affected CPs when the tasks change. Therefore, defining these share elements once and importing them into CPs foster reusability and flexibility of the modeling process.

Besides the above modeling capabilities, a DSL should offer the following features to support the modeling task:

SF1: The DSL should offer an IDE. For textual DSLs, the IDE should have syntax highlighting and auto-completion capabilities.

Rationale: Auto-completion, syntax highlighting and syntax checking can enhance the productivity when defining CP elements and increase usability (Cook et al., 2007, pp. 16-17; Merkle, 2010). In combination with syntax highlighting, the IDE can prevent syntax errors as 1) unusual color changes indicate an element violates syntactic rules, and 2) insert predefined templates or auto-completing keyword prevents typos in defining CP elements.

Remark: Acadela IDE does not auto-complete CP elements (e.g., identifier of Stages, Tasks, etc.).

SF2: The DSL should visualize the modeled CP using basic graphical notations to demonstrate the reconstructed elements, such as phases, control flows, activities and their inputs or outputs.

Rationale: A CP visualization provides visual clues that can enhance the understandability and readability of the models (Frank, 2010, p. 1). Another benefit is reducing the learning effort as the clinical experts do not have to understand modeling concepts in-depth (e.g., Hermans et al. (2009, p. 433)). Additionally, one noticeable feedback from interviews with technical staffs is that Acadela needs a graphical visualization to provide an overarching overview of the CP model. These visual clues assist in debugging and model analysis.

5.2 Language Specification

From the necessary CP modeling features described in the previous section, our study develops Acadela as a textual DSL to define CPs compatible with SACM. In addition, Acadela also aims to optimize the modeling process by providing 1) flexible syntactic rules, 2) automatic construction of default behaviors, and 3) concise constructs that combine schematic and UI display definitions. Our study derives these properties from guidelines of language design (Karsai et al., 2014; Frank, 2010). This chapter consecutively describes these DSL properties with examples to illustrate the optimization of the current modeling approach using XML.

5.2.1 Flexible Syntactic Rules

Since modelers in medical facilities have diverse programming knowledge, Acadela syntax should inherit syntactic features that promote flexibility and convenience from various existing programming languages. This practice can reduce the effort in designing language (Karsai et al., 2014, p. 3) and ease the CP definition task for modelers, as they can reuse syntactic rules similar to software design languages (e.g., Python, C, or SQL) (Frank, 2010, p. 13). As a result, Acadela possesses the following syntactic properties, which Figure 5.1 illustrates:

1. **Case-insensitive:** Modelers can write keywords in Acadela as UPPERCASE, CamelCase, or lowercase. This feature offers two benefits: 1) relieving users from syntax errors caused by wrong character casing, and 2) supporting different coding practices related to casing code elements. For example, `Task` and `Stage` can have uppercase or camel case, while their attributes are lowercase; or all keywords are uppercase or lowercase. Acadela gets inspiration from the case-insensitive property of SQL.
2. **Indent-insensitive:** Acadela does not assign any particular function to the tab character nor uses tabs to enforce a switch to a different hierarchical level. This property supports modelers in applying various indent strategies according to their coding practice. Acadela gets inspiration from indent-insensitive property of Java, Javascript, and C family programming languages.
3. **No Announcement Separator:** Acadela does not use a character to mark the end of a statement. This property is different from Java and C-family languages which use the semicolon. The purpose is to relieve users from ending their statements. Acadela gets inspiration of this feature from Python.

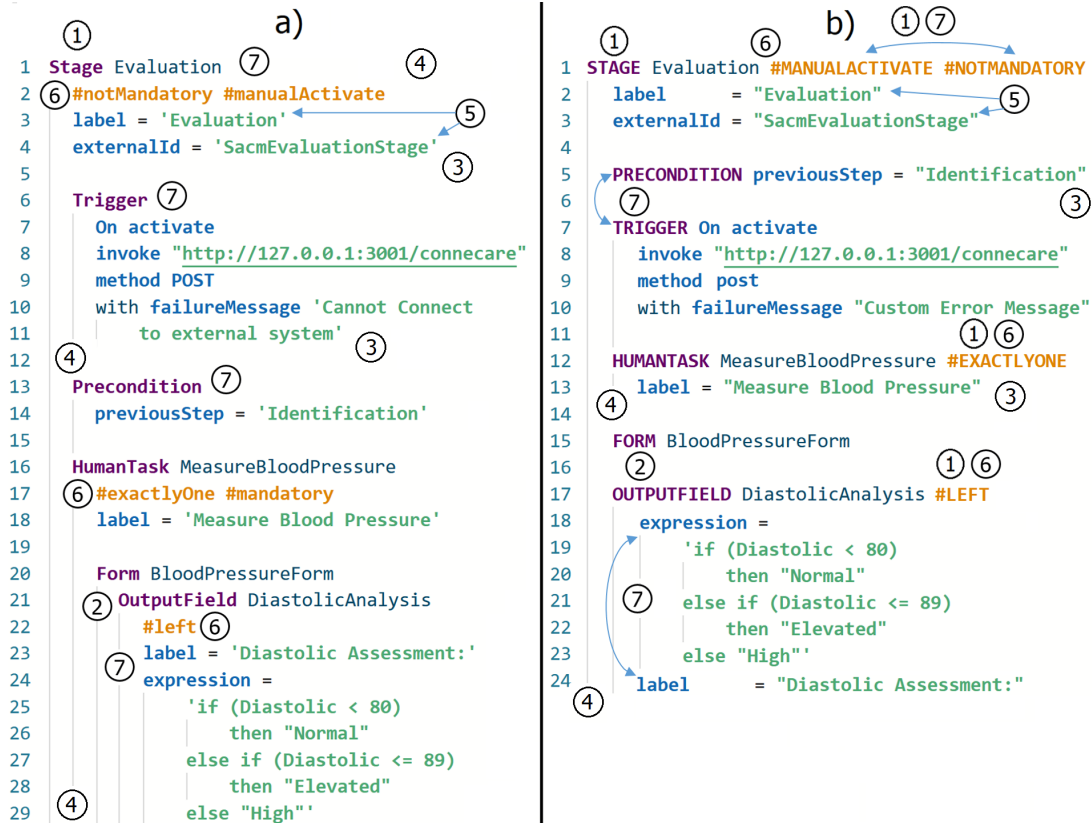


Figure 5.1: Illustration of Flexible Syntax Rules in Acadela. The two code snippets are syntactically different but semantically equivalent. The comparison demonstrates the Acadela properties of 1) Case-insensitive, 2) Indent-insensitive, 3) No Announcement Separator for each key-value assignment, 4) No End Indicator for each CP element declaration, 5) Interchangeable Quote and Double Quote for String, 6) Directives to represent system-defined attribute values, 7) Interchangeable Order of Directives, Child Elements, and Attributes

4. **No End Indicator:** Modelers do not have to type any keyword to mark the end of an element definition, as Acadela parses the legitimate attributes and child elements to construct an object. This property is different from SACM CaseDefinition model in XML, which requires a closing tag for every CP element.
5. **Interchangeable Quote and Double Quote for String:** Similar to many programming languages, the start and end of a String is a *single quote* or *double quote*. However, if modelers need to use quotes in the String value, they need to wrap the String using the other type of quote. For example, in Figure 5.1, the *uiReference* Attribute wraps the String with a single quote, but a pair of double quotes enclose the String value "None". Acadela gets inspiration from the same property of Javascript and Python.
6. **Directives:** For system-defined Attributes, i.e., Attributes that have a predefined set of possible values, their declaration starts with a hash (#) character followed by the Attribute value.

`#<attributeValue>`

For example, `#mandatory` directive applied to a Stage means the *isMandatory* attribute of the Stage is true, i.e., the Stage compulsory to complete a CP.

Acadela Directive gets inspiration of from Annotations of the Spring Boot framework.

7. **Interchangeable Order of Attributes and Child Elements:** For every CP element, Acadela syntax support the definition of Directives, Attributes, and eligible child CP elements. Therefore, modelers can arrange the items in each Directive, Attribute, or child CP elements group in any order. For example, Figure 5.1 shows that both code snippets declare a *non-mandatory* and *manually activated* Stage enabled only when the Identification Stage completes. However, the two code snippets have a different order of directives, attributes, and child CP elements.

Note that the declaration of Stages and Tasks follows a chronological order, i.e., the first Stage or Task appears first in the SACM UI, follow by the second Stage or Task. However, an exception occurs when a Stage or Task is *manually activated*. In this case, the Stage or Task is invisible until the caseworker manually triggers it.

5.2.2 Automatic Execution of Default Behaviors

Default Reference Assignment: Another default value type is the stereotypical reference path to an EntityDefinition of a Case element. In SACM, a Case, Stage, or Task contains an **entityAttachPath** stating where the newly instantiated Entity is attached (Michel, 2020, pp. 103, 107, 109). The starting point of **entityAttachPath** is the Case root Entity. The node in the path is the element ID, with each node separated by a dot. For example, the **entityAttachPath** to a *MeasureBmi* Task in an *Evaluation* Stage is *Evaluation.MeasureBmi*. As a result, the **entityAttachPath** of a Task follows the structure <StageID>.<TaskID>. While the **entityAttachPath** of a Case or Stage is its ID. For TaskParamDefinition, the typical reference **path** is the parent Stage ID, parent Task ID, and the AttributeDefinition ID of the parent Task separated by a dot:

<ParentStageID>.<ParentTaskID>.<AttributeDefinitionID>

Acadela constructs the **entityAttachPath** of each element based on their element type. For Case and Stage, Acadela assigns the Case and Stage ID to the *entityAttachPath*. While for Task and TaskParam elements, Acadela analyzes the parent Stage and Task to extracts their IDs for constructing the **entityAttachPath**.

In addition, Case, Stage, and Task elements also have an **entityDefinitionId**, which refers to their EntityDefinition (Michel, 2020, pp. 103, 107, 109). The ID of EntityDefinition can be the same with its corresponding CP element (e.g., StageDefinition, TaskDefinition). Therefore, Acadela defines the Case, Stage, or Task element with a unique ID. During the interpretation phase, Acadela generate the EntityDefinition of the CP element with the ID, and assign the ID as the **entityDefinitionId** value of the CP element. This unique ID is the shared property in the definition of the schema (EntityDefinition) and CaseDefinition, StageDefinition, or TaskDefinition element; hence Acadela can construct the two Definition objects of the same CP element using the unique ID.

Besides the **entityDefinitionId**, a CaseDefinition contains the **entityDefinitionId** and **entityAttachPath** of its Setting Entity. These attributes are named **newEntityDefinitionId** and **newEntityAttachPath** in the SACM Case object. The Case Setting stores the *case owner*, *case client* (patient), default *due date* definition, and *user-defined* Attributes. Acadela automatically assigns the Setting ID as the value of **newEntityDefinitionId** and **newEntityAttachPath** by default.

Figure 5.2 illustrates the default value assignment feature by presenting a StageDefinition in SACM syntax (picture 1) and the same StageDefinition written in Acadela syntax (picture 2). In

Acadela, modelers do not need to specify **entityDefinitionId** and **entityAttachPath** attributes, although they can manually declare their **entityDefinitionId** and **entityAttachPath**.

<p style="text-align: center;">(1)</p> <pre> 1 <StageDefinition id="GCS1_CaseIdentification" 2 description="Case Identification" 3 isMandatory="true" 4 repeatable="ONCE" 5 entityDefinitionId="GCS1_Identification" 6 entityAttachPath="GCS1_Identification"> 7 // Task Definitions 8 </StageDefinition></pre>	<p style="text-align: center;">(2)</p> <pre> 1 Stage CaseIdentification 2 label = "Case Identification" 3 // Task Definitions</pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------

Figure 5.2: Illustration of default constant value assignment in Acadela. In SACM (1), modelers need to declare the *mandatory* Attribute whether the Stage is *mandatory* or not. In Acadela (2), *mandatory* is the default value of the *mandatory* Attribute, and *repeatable* has *ONCE* as the default value, hence modelers do not need to specify it.

Default Constant Value Assignment: In the SACM CP definition, users frequently assign a specific value to certain attributes. For example, Figure 5.2 demonstrates that when a Stage is typically *mandatory*, the modelers need to specify the *isMandatory* Attribute in every StageDefinition of SACM (picture 1). However, in Acadela (picture 2), modelers do not need to declare the *mandatory* Directive as its default value is *mandatory*. Modelers can declare that the Stage or Task is not compulsory by adding a **#notmandatory** directive after the Stage or Task ID. Additionally, when Stages are usually *non-repeated*, Acadela sets the default value as *ONCE* to the *repeatable* Attribute; thus, modelers also do not need to declare it.

As a result, Acadela relieves the users from defining these *Attributes* by storing a dictionary with key-value pairs representing the *Attribute* name and its default value. The Interpreter section under the Implementation chapter discusses this feature in further detail (See Section 6.5). Consequently, modelers do not need to declare the default constant value of the *Attribute*. Instead, Acadela will add such attributes into SACM-compatible JSON objects of the CP element during the interpretation phase.

5.2.3 Concise Constructs

Single Clinical Pathway Element Definition: Defining a CP element in SACM requires the declaration of 1) a *schematic model* containing the element attributes and 2) the *behaviors* and *visualization* of the element in the CP execution. While this approach separates the *schemata*, *execution*, and *graphical representation* concerns of the element, its side effect is the duplication of attributes used by the model and visualization definitions. Furthermore, separating these concerns requires modelers to examine or modify two locations of a CP element when debugging or updating the element, which can be more error-prone than a centralized definition method, where modelers need to manage one definition for each CP element.

Therefore, Acadela optimizes the CP element definition by combining schema, case-related behaviors, and graphical representation into one object declaration. As a result, Acadela states the shared attributes of the three layers once. Afterward, during the interpretation phase, Acadela extracts the shared attributes and inserts them into the JSON objects of the schema and visualization definition of the CP element. In addition, Acadela includes all other schematic, case-related, visual representation properties, and child elements into a single CP element definition.

Figure 5.3 illustrates the combined approach of Acadela in defining a *Spirometry* measurement HumanTask in SACM. Picture (1) shows the a) schema definition of the Task and its input fields (EntityDefinition and AttributeDefinition); b) Case execution behaviors (HumanTaskDefinition), including the Precondition (SACM SentryDefinition) to activate this Task, which states the Stage or Task that must already be completed); and c) the features of the spirometric InputFields (TaskParamDefinitions), including their graphical representation (position attributes at line 29 and 34).

Picture (2) shows a combined definition of the HumanTask schema, behaviors, and visualization in Acadela. The *ID* and *description* in the EntityDefinition and HumanTaskDefinition converge into the Acadela HumanTask *ID* and *label*. In the interpretation phase, Acadela extracts these two properties to construct the EntityDefinition and HumanTaskDefinition JSON objects in SACM format. The Acadela InputField merges the TaskParam schema (AttributeDefinitions), case-related and graphical properties. The Form object contains *mandatory* and *readonly* attributes that apply to all fields; hence modelers do not need to specify these two properties for every TaskParam. Similar to SACM, other child elements of the Task, like the Precondition (SentryDefinition in SACM), also reside within the Task Definition.

(1)	(2)
<pre> 1 <EntityDefinition id="GCS1_Spirometry" 2 description="Lung función - Spirometry"> 3 4 <AttributeDefinition id="spiro1" 5 type="number" multiplicity="exactlyOne" 6 description="FEV1 post in Liters" /> 7 8 <AttributeDefinition id="spiro2" 9 type="number" multiplicity="exactlyOne" 10 description="FEV1 post in %" /> 11 </EntityDefinition> 12 13 <HumanTaskDefinition id="GCS1_Spirometry" 14 isMandatory="false" 15 repeatable="ONCE" 16 description="Spirometry" 17 ownerPath="GCS1_Settings.Clinician" 18 dueDatePath="GCS1_Settings.EvaluationDueDate" 19 entityDefinitionId="GCS1_Spirometry" 20 entityAttachPath="GCS1_Evaluation.GCS1_Spirometry"> 21 22 <SentryDefinition> 23 <precondition 24 processDefinitionId="GCS1_SetEvaluationDueDate"/> 25 </SentryDefinition> 26 27 <TaskParamDefinition 28 isReadOnly="false" isMandatory="false" 29 position="LEFT" 30 path="GCS1_Evaluation.GCS1_Spirometry.spiro1"/> 31 32 <TaskParamDefinition 33 isReadOnly="false" isMandatory="false" 34 position="LEFT" 35 path="GCS1_Evaluation.GCS1_Spirometry.spiro2"/> 36 </HumanTaskDefinition> </pre>	<pre> 1 HumanTask Spirometry 2 #notMandatory 3 owner = 'Setting.Clinician' 4 label = 'Spirometry' 5 dueDateRef = 'Setting.EvaluationDueDate' 6 7 Precondition 8 previousStep = 'SetEvaluationDueDate' 9 10 Form SpirometryForm 11 #notReadOnly #notMandatory 12 13 InputField spiro1 14 #number #exactlyOne 15 label = 'FEV1 post in Liters' 16 17 InputField spiro2 18 #number #exactlyOne 19 label = 'FEV1 post in %' </pre>

Figure 5.3: Illustration of Acadela approach to combine *schema*, *execution behaviors* and *visual representation* in one single CP element definition. The HumanTask definition in SACM (1) requires the schema definition (EntityDefinition) and case-related execution and visualization (HumanTaskDefinition). Meanwhile Acadela (2) groups all HumanTask attributes, behaviors, and graphical representation into a single element.

Hierarchical Declaration of Clinical Pathway Elements: Acadela arranges CP elements in a subsumption order to provide a hierarchical structure of CP models. For example, a Case is the master object that contains multiple Stages, and each Stage consists of Tasks; thus, Acadela enforces a subsumption rule to declare all Stages elements within the Case object and define Tasks within its Stage. Figure 5.4 illustrates the hierarchy of CP elements in Acadela. Note that

the color code in Figure 5.4 is to distinguish CP elements in Acadela. The color code does **not** represent the architecture layer described in Figure 4.3.

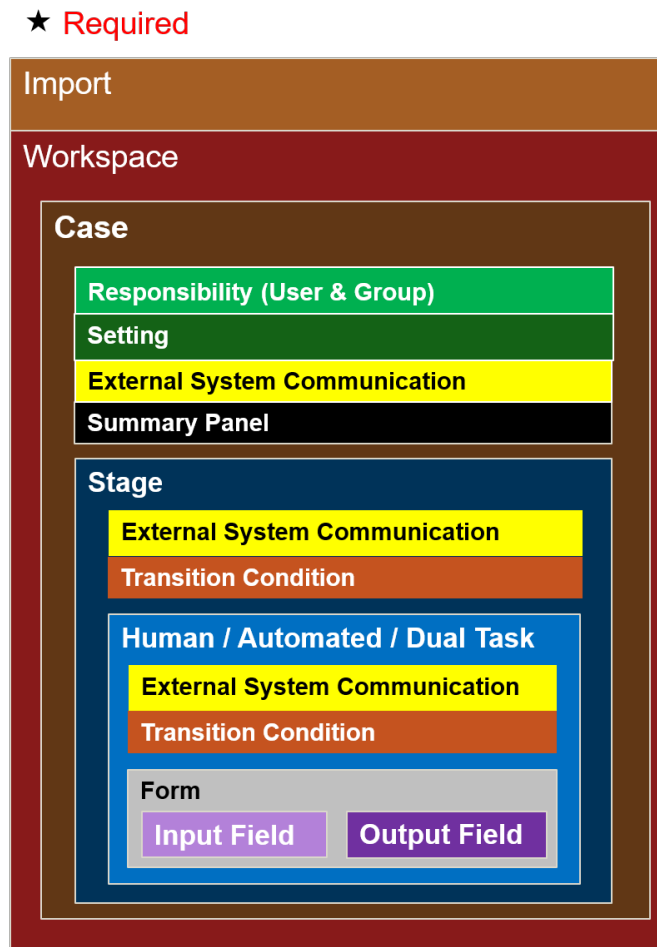


Figure 5.4: Hierarchy of elements in a clinical pathway model expressed in Acadela.

The next section describes each CP element in further detail. The remainder of this subsection presents an overview of the CP elements introduced in Figure 5.4.

Acadela begins the CP model definition with an optional `Import` section, in which modelers can include CP elements declared in other files. Next, the `Workspace` section states the repository where the `Case` definition will be stored. Therefore, `Workspace` usually specifies the name of a medical facility that applies the CP. Additionally, the `Workspace` contains the `Case` object, which defines the CP for treating a particular disease.

The `Case Responsibilities` element includes references to 1) medical teams or professionals involved in the treatment and 2) a *patient group* to enable the search of a patient in the group. The `Case Setting` includes 1) the *case owner*, 2) *global variables* accessible to the `Case` elements, e.g., the *default due date* used by `Stages` or `Tasks`, and 3) the selected medical teams to participate in the treatment. *External System Communication* is an optional element that specifies how to send the data of a `Case`, `Stage`, or `Task` to an external system. `Summary Panel` declares interesting medical data from the `Case` elements and their grid layout position in the UI. Finally, the `Stage` section defines all the treatment phases.

Stage and Task contain *External System Communication* and *Transition Condition(s)*. The latter specifies 1) the previous Stage or Task that must be completed to trigger the current one and 2) an optional condition to activate the Stage or Task.

A Task has a Form containing optional global *readonly* and *mandatory* attributes applied to all of its InputFields. The InputField is an element to collect medical data, and OutputField expresses how to compute or display specific information based on the inputs.

To establish consistency in the model code, Acadela restricts CP elements of the same type to be declared consecutively. For instance, Responsibilities (Groups and Users) are at the same hierarchical level as Stages, but modelers cannot declare the Responsibilities element in the middle of the two Stages. Instead, the correct position of Responsibilities is before or after the declaration of all Stages.

As described in the flexible syntax rules, the order of children objects in a CP element is interchangeable. As a result, modelers can declare Responsibilities, Setting, *External System Communication*, SummaryPanel, and Stage in any sequence. For example, modelers can swap the position of the Stages definition to Responsibilities. This feature helps medical software development teams to arrange elements flexibly according to their standards or preferences.

5.3 Concrete Syntax for Clinical Pathway Element Definition

This section demonstrates how Acadela declares CP elements in SACM by leveraging its language properties described in the above section. Every CP element specification has a table listing all attributes, their features, and how to express them (concrete syntax) in Acadela. Additionally, each element presents its minimum, and complete versions declared in SACM and Acadela syntax. This presentation aims to illustrate how Acadela optimizes CP model definitions in SACM. The Grammar section in the Implementation chapter explains how the Acadela grammar (abstract syntax) shapes the concrete syntax.

Since SACM reuses specific Attributes in multiple CP elements, this section first explains these shared Attributes, their applicable CP elements, and expression in SACM and Acadela.

The color code in the tables indicates the architectural layer, as expressed in Figure 4.3.

5.3.1 Data Type

SACM defines a **type** Attribute containing various 1) primitive data types used to validate the value of an InputField (See 5.3.3) or enforce the type of an OutputField (See 5.3.4) and 2) SACM built-in data types applicable to a specific CP element. (Michel, 2020, p. 101)

By default, SACM sets the type value as *notype*.

Data types marked with ^A symbol is built-in Acadela types to combine multiple types or express the attribute intuitively.

Data Type	Expression in Acadela	Description
notype	#notype	No data type applied to this field.
string	#string	A text type contains characters, symbols or numbers.
text ^A	#text	Same as string, but text is an alternative understandable to medical professionals.

5 Language Design

longtext	<code>#longtext</code>	Display a text area in the UI to show multi-line text.
boolean	<code>#boolean</code>	Accepts true or false value.
number	<code>#number</code> <code>#number(MIN-MAX)</code> <code>#number([Sign][Number])</code> where [Sign] in { =, <>, <=, >=, <, > }	Represent a numeric value: <ul style="list-style-type: none"> • <code>#number</code>: any number without restriction • <code>#number(MIN-MAX)</code>: accepts a number within the MIN and MAX range. For example, <code>#number(1-10)</code> means the InputField only accepts numbers from 1 to 10. • <code>#number([Sign][Number])</code>: accepts a number according to the comparison expression. For instance, <code>#number(>100)</code> only accepts any number larger than 100.
singlechoice ^A	<code>#singlechoice</code>	Indicates a multiple-choice question that only accepts a single answer among the given options. Acadela interprets this type as an Enumeration object when compiling to SACM format.
multiplechoice ^A	<code>#multiplechoice</code>	Indicates a multiple-choice question that accepts multiple answers among the given options. Acadela interprets the InputField type as Enumeration and the <i>multiplicity</i> Attribute as atLeastOne when compiling to SACM format.
date	<code>#date</code> <code>#date.after(TODAY)</code>	State a date type. <code>#date.after(TODAY)</code> sets the date value 24 hours from the current time.
json	<code>#json</code>	Indicates a JSON object. This type is useful for showing chart data expressing each data point as a JSON entity.
custom ^A	<code>#custom(path)</code>	If applied to an InputField, Acadela assigns the value from the path variable to the InputField. For OutputField, Acadela stores the Output into the element at the <i>path</i> location. The path variable points to a CP element in the Case starting from the Case level, For example, <i>Setting.BloodPressure</i> points to the <i>BloodPressure</i> Attribute in the Case Setting.
link	<code>#link.Users(UserOrGroup)</code> <code>#link.Entity(EntityId)</code>	<code>#link.Users</code> declares references to any User or Group stated in the Case Responsibilities (See 5.3.11). Meanwhile, <code>#link.Entity</code> declares a reference to any CP element (e.g., Stage, Task, InputField, or OutputField) through its ID.
documentLink ^A	<code>#documentLink(Url)</code>	A shortcut made by Acadela to insert a link to an external document accessible from the URL. The InputField value in the UI of SACM is the URL link. SACM creates the link to the document by setting the <i>dataType</i> to <i>string</i> , <i>readOnly</i> and <i>mandatory</i> Attributes to <i>false</i> , <i>uiReference</i> to <i>privatelink</i> , and <i>defaultValue</i> is the document's URL. Acadela sets all these properties accordingly when interpreting an InputField with <i>documentLink</i> type, so the InputField is compatible with SACM.

Table 5.2: Description of Data Types in Acadela based on SACM specification. (Michel, 2020, p. 101)

5.3.2 Mandatory Attribute

SACM uses the *isMandatory* Attribute in a Stage or Task to specify that it is compulsory to accomplish the Case or parent Stage, respectively. The *isMandatory* Attribute applied to an InputField or OutputField specifies that the field must have a valid value in order to complete the Task.

In SACM, *isMandatory* is a boolean Attribute having a *true* or *false* value (Michel, 2020, p. 106). Acadela expresses the true and false values using **#mandatory** and **#notMandatory**, respectively. To offer convenience for modelers, Acadela sets *isMandatory* to true by default, so modelers only specify the *isMandatory* Attribute when the Stage, Task, or Input/OutputField is not compulsory for accomplishing a CP. Table 5.3 shows an example to declare the *isMandatory* Attribute of a Stage in SACM and Acadela.

SACM	Acadela
<pre><StageDefinition id="Identification" isMandatory="true" ...> </StageDefinition></pre>	<pre>Stage Identification #mandatory</pre>
<pre><StageDefinition id="Identification" isMandatory="false" ...> </StageDefinition></pre>	<pre>Stage Identification #notMandatory</pre>

Table 5.3: An example of expressing the *isMandatory* Attribute of a Stage in SACM and Acadela.

5.3.3 Input Field

This element presents a **graphical representation** to collect **medical data** related to an activity conducted by **healthcare professionals**. An Acadela InputField is a combination of SACM TaskParamDefinition and AttributeDefinition. For this reason, Acadela supports the declaration of 1) *layout position*; 2) *visual effect* (e.g., painting background colors according to the input values); 3) the accepted *data type* (e.g., number, text, or multiple-choice question) of the InputField. The data type determines how SACM renders the InputField in the UI, i.e., displaying a field for the text or number type or showing the question and radio button or checkbox for *single-choice* or *multiple-choice* questions; 4) *readonly* to state that the InputField value cannot be changed; 5) whether the field is *mandatory* for completing the Task. **Note that SACM does not allow an InputField to be both *mandatory* and *readonly*.**

The Acadela InputField combines AttributeDefinition, EnumerationOption (for declaring multiple-choice questions), and TaskParamDefinition of SACM. (Michel, 2020, p. 101, 102, 112). Table 5.4 shows the SACM Attributes incorporated in the Acadela InputField element. The color code applied to each SACM Attribute indicates its SACM architecture layer.

SACM Attribute	Acadela Expression	Description
<code>id*</code>	<code><ID></code>	The identifier used as a reference within the Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is " <i>name</i> " as textX uses the " <i>name</i> " property as a reference to identify an imported element in a file.
<code>description*</code>	<code>label = "<text>"</code>	A text displayed in the UI which describes the input to care professionals. Acadela replaces the description keyword with the label term to emphasize the UI rendering characteristic of this attribute. This description can be expressed using the local human language.
<code>type*</code>	<code>#<type></code> (See Section 5.3.1)	Enforce a data type constraint (e.g., text, number, multiple-choice question, link to a CP element) on the InputField. See Section 5.3.1 for a detailed description of type in Acadela.

5 Language Design

multiplicity	<p>#atLeastOne #exactOne (default) #maximalOne #any</p>	<p>Declares the expected quantity of value in the InputField. For example, a multiple-answer multiple-choice question can have #atLeastOne or #any multiplicity as more than one answer can be added to the InputField value. Available <i>multiplicity</i> options are #atLeastOne, #exactOne, #maximalOne, or #any.</p>
additionalDescription	<p>additionalDescription = "<text>"</p>	<p>a text expresses explanatory information related to the current InputField. When declaring the <i>additionalDescription</i> value, SACM creates a question mark icon next to the InputField. hovering over this icon displays the <i>additionalDescription</i> content as a tooltip text.</p>
uiReference	<p>uiRef = "<text>"</p>	<p>Express the UI effect applied to this InputField. For example, define a background color band for ranges of numeric values, or display the InputField as a URL link to an external file.</p>
question ^A	<p>question = "<text>" option "<option1>" value = "<value1>" externalId = "<text>" additionalDescription = "<text>"</p> <p>option "<option2>" value = "<value2>" externalId = "<text>" additionalDescription = "<text>"</p>	<p>Declares a multiple-choice question with options. Each option has a value attribute, which SACM assigns to the value of the InputField when the option is selected. In addition, modelers can specify the type as #singlechoice or #multiplechoice to indicate whether one or multiple answers are accepted. <i>externalId</i> and <i>additionalDescription</i> attributes are optional and function like the ones of the InputField. The <i>multiple-choice question</i> is a compressed syntax for declaring an EnumerationOption in SACM.</p>
externalId	<p>externalId = "<text>"</p>	<p>Declares an ID to map with an external system</p>
defaultValues	<p>defaultValues = "<text>"</p>	<p>Declares a list of multiple values that SACM will initially set to the InputField</p>
defaultValue	<p>defaultValue = "<text>"</p>	<p>Declares a single value that SACM will initially set to the InputField</p>
path	<p>For custom path to an element, set directive to #custom^A and declare: <i>ElementPath</i> = "<pathToElement>"^A</p>	<p>By default, Acadela declares a path pointing to the InputField Entity from the parent Stage. Therefore, the default path value is <StageId>.<TaskId>.<InputFieldId>. For example, the path to an InputField Age in the Task <i>AdmitPatient</i> of the Stage <i>Identification</i> is <i>Identification.AdmitPatient.Age</i>. Modelers do not need to specify the default path. However, a custom path pointing to a different CP element must be declarable with a #custom and an <i>ElementPath</i> attribute containing the reference path to a CP element.</p>
isMandatory	<p>#mandatory (default) #notMandatory</p>	<p>States whether the InputField must contain a value or not to complete a Task. See Section 5.3.2 for further information.</p>

5 Language Design

isReadOnly	#readOnly #notReadOnly (default)	States whether the InputField displays its value in a readonly format or expects a user inputs. The default value is false as medical tasks frequently need to collect medical data.
position	#left #leftcenter #center #centerright #right #stretched (default)	States the grid layout position to render the InputField in the UI, which contains three grid columns. The supported values are: #left: the left grid cell #center: the middle grid cell #right: the right grid cell #leftcenter: span across the first two cells #centerright: span across the last two cells #stretched: span across all the cells
part	#humanDuty #systemDuty	Only applicable to InputFields of a DualTask. This Attribute states whether a case worker or a system inputs data into the InputField.

Table 5.4: The attributes of an Acadela InputField.

* - the attribute is required.

A - the attribute is created in Acadela and does not exist in SACM.

The following examples illustrate the declaration of the InputField in different scenarios. First, Listing 5.1 and 5.2 show the minimum and complete versions of the InputField declaration. The minimum version contains only the compulsory InputField attributes. The complete version shows all declarable attributes in the InputField. Figure 5.5 shows the UI displayed in SACM for the InputField declared in Listing 5.1.

```

1 InputField FIELDNAME
2 #text
3 label = 'label'

```

Listing 5.1: Minimum Input Field Declaration

label *

Figure 5.5: Input Field Rendering in SACM produced by Listing 5.1

```

1 InputField FIELDNAME
2 #mandatory
3 #notReadOnly
4 #left
5 #maxOne
6 // #humanDuty // for Dual Task
7 #text
8
9 label = 'label'
10 additionalDescription = 'extraInfo'
11 // for custom reference to a CP object
12 // CustomFieldValue = '
13     pathToCaseObject'
14 uiRef = 'visualizationDefinition'
15 externalId = 'IdInExternalSystem'
16 defaultValue = 'defaultValue'
17 //defaultValues = ['val1', 'val2']

```

Listing 5.2: Complete Input Field Declaration

The second example illustrates how to define a *multiple-choice question* that accepts a single value (Listing 5.3) or multiple values (Listing 5.4). Figure 5.6 and 5.7 show the graphical representations of the InputField in both question modes.

```

1 InputField ApplyHeat
2 #singleChoice
3 question =
4 "Using Heat Lamp?"
5 option "No" value = '0'
6 option "Yes" value = '1'

```

Listing 5.3: Multiple Choice Question - Single Answer

```

1 InputField AcupuncturePos
2 #multiplechoice
3 question = 'Message Positions:'
4 Option "Temple"
5 value = 'TEMPLE'
6 Option "Shoulder"
7 value = 'SHOULDER'
8 Option "Jaw"
9 value = 'JAW'

```

Listing 5.4: Multiple Choice Question - Multiple Answers

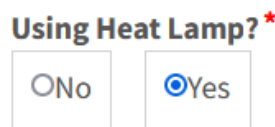


Figure 5.6: Rendering of the Multiple Choice - Single Answer InputField in the SACM UI. The InputField value in this example is '1'.

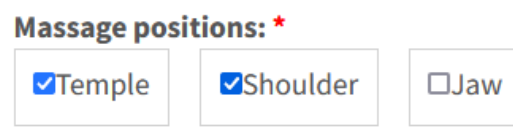


Figure 5.7: Rendering of the Multiple Choice - Multiple Answer InputField in the SACM UI. The InputField value in this example is ['TEMPLE', 'SHOULDER'].

5.3.4 Output Field

An Acadela OutputField supports the following capabilities to compute and display an output from InputField(s):

1. **Mathematical operations:** SACM supports basic arithmetic and square root calculations. Additionally, Acadela automatically converts a string to a number in the formula using the *number(<InputFields>)* function provided by SACM.
2. **Conditional expressions:** produce outputs based on conditional statements applied to InputField(s). Compound and complex conditional expressions are declarable using AND or OR operators and parentheses. Note that the OutputField only accepts Input/OutputField(s) ID within the same Task; thus, referencing an Input/OutputField outside the OutputField's Task is not supported.
3. **Background color coding:** the *uiReference* attribute stores definitions of a color band applied to the OutputField's result. Therefore, modelers can declare a background color applied to a range of numeric values, e.g., if the output Body Mass Index (BMI) value is from 18.5 to 24.9 (normal), display a green background to the BMI OutputField.
4. **Dynamic template rendering:** display a custom visual effect on a SVG image based on the InputField(s) value. To enable this feature, first modelers define an InputField storing an SVG template that contains a medical image and custom graphical content (e.g., painted circle, rectangle containing text data). Next, SACM requires another OutputField that returns the template style based on the InputField(s) value. The OutputField uses SACM conditional expression syntax to generate the style based on the corresponding conditions applied to InputField(s).

Table 5.5 demonstrates the attributes of an OutputField in SACM and the color code to express their hierarchy level in the Architecture of SACM.

5 Language Design

SACM Attribute	Acadela Expression	Description
id*	<ID>	The identifier used as a reference within the Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is "name" as textX uses the "name" property as a reference to identify an imported element in a file.
description*	label = "<text>"	A text displayed in the UI which describes the output to care professionals. Acadela replaces the description keyword with the <i>label</i> term to emphasize the UI rendering characteristic of this Attribute. This description can be expressed using the local human language.
explicitAttributeType*	#<type> default: #notype (See Section 5.3.1)	Enforce an data type constraint (e.g., text, number) on the OutputField. See Section 5.3.1 for a detailed description of type in Acadela.
additionalDescription	additionalDescription = "<text>"	a text expresses explanatory information related to the current OutputField. When declaring the <i>additionalDescription</i> value, SACM creates a question mark icon next to the OutputField. When hovering over this icon, the UI displays the <i>additionalDescription</i> as a tooltip text.
uiReference	uiRef = "<text>"	Express the UI effect applied to this OutputField. For example, define a background color band for different ranges of numeric value (See Listing 5.6 and Figure 5.9), show the data as a line diagram (uiRef = "linediagram"), or display an SVG of the medical data (uiRef = "svg").
expression	uiRef = "<text>"	Define how to render an output value using conditional expression, mathematical formula, graphical effects (e.g., CSS style definition), or SVG template that involves InputField(s). Listing 5.5, 5.6, and A.9 respectively show the syntax for the above rendering mechanisms.
externalId	externalId = "<text>"	Declares an ID to map with an external system

path	For custom path to an element: set directive to <code>#custom</code> and <code>ElementPath = "<pathToElement>"</code> ^A	Declares a path pointing to the Entity of the other Case object that displays the OutputField value. Acadela automatically generates the path to the OutputField as follows. If the path points to an InputField or OutputField of a Stage, then the path value is <code><StageId>.<TaskId>.<InputFieldId></code> . For example, the path to an InputField <i>Age</i> in the Task <i>AdmitPatient</i> of the Stage <i>Identification</i> is <code>Identification.AdmitPatient.Age</code> . If the path points to a Setting Attribute, then the path is <code>Settings.<AttributeName></code>
isMandatory *	<code>#mandatory</code> <code>#notMandatory</code> (default)	States whether the OutputField must contain a value or not to complete a Task. See Section 5.3.2 for further information.
isReadOnly	<code>#readOnly</code> <code>#notReadOnly</code> (default)	States whether the OutputField value is overridable when modifying the Task data. However, in SACM, the result of an OutputField is not modifiable as it is generated from the value of InputField(s)
position	<code>#left</code> <code>#leftcenter</code> <code>#center</code> <code>#centerright</code> <code>#right</code> <code>#stretched</code> (default)	States the grid layout position to render the OutputField in the UI, which contains 3 grid columns. The supported values are: <code>#left</code> : the left grid cell <code>#center</code> : the middle grid cell <code>#right</code> : the right grid cell <code>#leftcenter</code> : span across the first two cells <code>#centerright</code> : span across the last two cells <code>#stretched</code> : span across all the cells

Table 5.5: The attribute of an Acadela OutputField.

* - the attribute is required.

A - the attribute is created in Acadela and does not exist in SACM.

Conditional Output Expression: Listing 5.5 illustrates an example of declaring a minimum *Blood-PressureCondition* OutputField that conditionally displays the result. In blood pressure measurement (United Kingdom National Health Service, 2021), a Systolic/Diastolic blood pressure value below 90/60 mmHg is *low*, from 90/60 to 120/80 mmHg is *healthy*, 120/80 to 140/90 mmHg is *pre-high*, and above 140/90 mmHg is *high*. Figure 5.8 shows the SACM output of the blood pressure status based on the code of Listing 5.5.

Dynamic Background Color Rendering: Listing 5.6 and Figure 5.9 show how to define a color band for rendering background color according to a numeric value range. In this example, yellow, green, orange, and red color codes apply to the underweight, normal, overweight, and obese BMI scale. Since the height and weight value is normal, SACM renders a green background to the OutputField.

Note that in SACM the same color cannot be used twice in a *uiReference* definition. Additionally, decimal values (e.g., 18.5) deactivate the coloring function.

```

1 InputField Systolic
2   #number(0-300)
3   label = 'Systolic Blood Pressure:'
4
5 InputField Diastolic
6   #number(0-300)
7   label = 'Diastolic Blood Pressure:'
8
9 OutputField BloodPressureCondition
10  #left
11  label = 'Blood Pressure Assessment:'
12  expression = '
13  if (Systolic >= 140 or Diastolic >= 90)
14  then "High"
15  else if
16  (Systolic > 120 or Diastolic > 80)
17  then "Elevated"
18  else if
19  (Systolic > 90 or Diastolic > 60)
20  then "Normal"
21  else "Low"

```

Listing 5.5: Minimum OutputField Declaration of a conditional output. The BloodPressureCondition OutputField shows the blood pressure status based on the Systolic and Diastolic values.

Systolic Blood pressure (mm Hg):

Diastolic Blood pressure (mm Hg):

Blood Pressure Assessment:

High

Figure 5.8: SACM Display of the Conditional OutputField in Listing 5.5. Here the blood pressure is high because the Diastolic value is above 90.

```

1 InputField Height
2   #number(0-3) #exactlyOne
3   label = 'Height (m)'
4
5 InputField Weight
6   #number(0-500) #exactlyOne
7   label = 'Weight (kg)'
8
9 OutputField BmiScore
10  #left
11  label = 'BMI Score:'
12  uiRef = 'colors(0 < yellow < 18 <
13         green < 25 < orange < 30 < red <
14         100)'
15  expression = 'Weight / (Height *
16              Height)'

```

Listing 5.6: Code Definition of color codes for ranges of numeric values using the uiRef Acadela attribute.

Height of patient in m *

Weight of patient in kg *

BMI Score: * 22

Figure 5.9: Background Color Effect from the color code definition to the BmiValue OutputField in Listing 5.6.

Dynamic Visualization : The following example demonstrates the definition of a customizable UI template to show the head massage positions and their affected meridian vessels (Listing 5.10). Acadela requires the OutputField to contain 1) an expression of an SVG image that helps modelers defines region and symbols in a graphical image (lines 9-28); 2) a SACM command to dynamically define the CSS styles based on InputField(s) value (lines 30-41). Examples of styles are hiding or showing elements or drawing a fore- or background color; 3) a CSS style placeholder (line 18) in the SVG to be replaced by the CSS style definition in 2) (lines 43-46). In the following example, a massage position from an InputField will add a style to fill circles having the same body part as the class name with blue, thus creating the dynamic rendering image as Figure 5.10 illustrates.

5 Language Design

```

1 InputField HeadMessagePosition #multipleChoice
2   question = "Message the following positions:"
3   option "Temple" value = "TEMPLE"
4   option "Nape" value = 'NAPE'
5   option "Jaw" value = 'JAW'
6   // ... - Options for other head or body areas
7   option "Forehead" value = "FOREHEAD"
8
9 InputField messageLocationTemplate #string #exactlyOne // Import SVG Template
10  label = "Meridian Template"
11  uiRef = 'hidden'
12  defaultValue = '<svg version="1.0" xmlns="http://www.w3.org/2000/svg"
13    xmlns:xlink="http://www.w3.org/1999/xlink"
14    width="<width>pt" height="<height>pt"
15    viewBox="0 0 <width> <height>"
16    preserveAspectRatio="xMidYMid meet">
17
18    <style> #dynamicstylevars{} </style> // style placeholder
19    <image xlink:href=<linkToImagePath> />
20    // use <g ...> <path d="<data points>" /> </g> for custom SVG graphic
21
22    <circle class="temple" cx="86" cy="60" r="3" fill="none"/>
23    <circle class="temple" cx="122" cy="60" r="3" fill="none"/>
24
25    <circle class="nape" cx="492" cy="84" r="3" fill="none"/>
26    <circle class="nape" cx="509" cy="84" r="3" fill="none"/>
27
28    // ... - Other custom shapes and location definition'
29
30 OutputField messageLocationStyle #string // Dynamic Style based on inputs
31  label = "Message Style"
32  uiRef = "hidden"
33  expression = 'let messageSites = HeadMessagePosition in
34    let styleTemple = if messageSites.contains("TEMPLE")
35      then ".temple{fill:blue}" else "" in
36    let styleNape = styleTemple + if messageSites.contains("NAPE")
37      then ".nape{fill:blue}" else "" in
38    // ... - code to define style for each body part
39    let styleForehead = styleNeck + if messageSites.contains("FOREHEAD")
40      then ".forehead{fill:blue}"
41    else "" in styleForehead'
42
43 OutputField messageLocationVisual #string
44  label = "Potential Message Points" uiRef = 'svg'
45  // Replace the placeholder in SVG with the dynamic style
46  expression = 'replace(messageLocationTemplate, "#dynamicstylevars{}",
    messageLocationStyle)'

```

Listing 5.7: Dynamic Template Rendering Definition code in Acedela

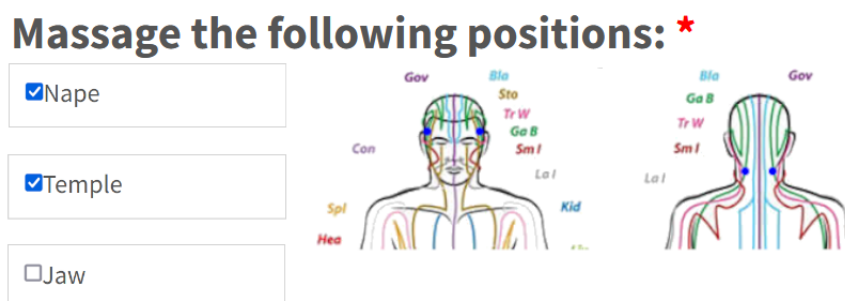


Figure 5.10: Illustration of InputField values (message positions - left) affect the output image visualization (right) by showing temple and nape message positions as blue circles.

5.3.5 Form

The Form is an Acadela element that contains all InputFields and OutputFields of a Task. Each Form has an ID to enable importing Form from an external file by ID. Forms support global *mandatory* and *readOnly* directives that uniformly declare the two attributes to all InputFields and OutputFields during the interpretation phase. Table 5.6 describes the value of the two attributes. Modelers can also specify a different *mandatory* or *readOnly* Attribute value to a particular field to override the global directives. This feature reduces repetitive definition of the two attributes in multiple InputFields or OutputFields. Listing 5.8 and 5.9 respectively show the Form declaration with and without global directives.

SACM Attribute	Acadela Expression	Description
isMandatory	<code>#mandatory</code> (default) <code>#notMandatory</code>	Declares whether all InputField OutputField must contain a value (<code>#mandatory</code>) or not (<code>#notMandatory</code>) to complete a Task. See Section 5.3.2 for further information.
isReadOnly	<code>#readOnly</code> <code>#notReadOnly</code> (default)	Declare whether all InputField or OutputField value is overridable when modifying the <i>Task</i> data. However, in SACM, the result of an OutputField is not modifiable as it is generated from the value of InputField(s)

Table 5.6: The Acadela Form attributes apply to all the included InputFields and OutputFields.

```

1 Form BmiForm #readOnly
2
3   InputField Height #number(0-3)
4     label = 'Height (m)'
5
6   InputField Weight #number(0-500)
7     label = 'Weight (kg)'
8
9   OutputField BmiScore #number
10     label = 'BMI Score:'
11     uiRef = '<colorCode>'
12     expression =
13       'Weight / (Height * Height)'
14
15   InputField DoctorNote
16     #text #notReadOnly
17     label = "Note:"

```

Listing 5.8: Form Definition example with global `readOnly` attributes applied to all InputFields and OutputFields except the DoctorNote InputField, which has a local `notReadOnly` directive

```

1 Form BmiForm
2   InputField Height
3     #number(0-3) #readOnly
4     label = 'Height (m)'
5
6   InputField Weight
7     #number(0-500) #readOnly
8     label = 'Weight (kg)'
9
10  OutputField BmiScore
11    #stretched #number #readOnly
12    label = 'BMI Score:'
13    uiRef = '<colorCode>'
14    expression =
15      'Weight / (Height * Height)'
16
17  InputField DoctorNote
18    #text #notReadOnly
19    label = "Note:"

```

Listing 5.9: Form Definition without global attribute. Each field specifies its own `readOnly` attribute

5.3.6 Precondition

Acadela Precondition is a condensed version of SentryDefinition in SACM to define *control flows*. The Precondition *previousStep* attribute states the ID of the prerequisite Stage or Task to trigger the current one. The *condition* Attribute declares the criteria needed to trigger the current Stage or Task. The *condition*'s boolean expression can include an InputField, OutputField, or number of Stage's iterations. A Stage or Task can have many Precondition definitions. This ability enables the definition of complex

5 Language Design

decision criteria to trigger alternative execution paths, i.e., activate a particular Stage or Task based on the current medical status. Listing 5.10 demonstrates a decision criteria written as a complex boolean expression that controls the condition to activate a Stage.

Additionally, Acadela supports the declaration of multiple Preconditions to display a disjunction (OR) relationship among them. SACM expresses a Conjunction relationship (AND) of prerequisite Stages or Tasks by declaring multiple *previousStep* attributes in a Precondition. Listing 5.10 and Listing 5.3.6 show how to declare the disjunction and conjunction relationships in Acadela, respectively.

Furthermore, SACM creates a repetitive Stage by assigning the current Stage ID to the *previousStep* attribute of the same Stage's Precondition, as illustrated in Listing 5.15 and Figure 5.14 in Section 5.3.9.

SACM Attribute	Acadela Expression	Description
<code>processDefinitionId*</code>	<code>previousStep = "<StageOrTaskID>"</code>	Declares the prerequisite Stage or Task that must be completed to trigger the current Stage or Task.
<code>expression</code>	<code>condition = "<booleanExpression>"</code>	Declare the decision criteria as a boolean expression to activate the current Stage or Task. SACM supports the definition of <i>compound</i> and <i>complex boolean expressions</i> .

Table 5.7: The attributes of Acadela Precondition.
* - the attribute is required.

```

1 Stage Identification
2   label = 'Identification'
3
4   HumanTask SelectPatient
5     label = 'Assign Patient'
6
7     Form PatientAssignForm
8       InputField PatientAge #number
9         label = "Patient age"
10
11    Stage CY #repeatserial
12      label = 'Cytological-Testing'
13
14      Precondition previousStep = 'Identification'
15      Precondition previousStep = 'CY'
16        condition = '(CY.AssessCY.Colp = 2 or CY.AssessCY.Colp = 3)
17                    and (Identification.SelectPatient.PatientAge < 30)'
18
19      HumanTask AssessCY
20        label = 'Evaluate test results'
21
22      Form CYForm
23        InputField Colp #singlechoice
24          question = 'Cytological testing'
25            Option 'Pap II-p,g' value='2'
26            Option 'Pap IIID-1' value='3'
27          // Other Options

```

Listing 5.10: Declaration of repetitive Stage with complex conditional expression. The CY Stage in this example is repeated if the cytological test result obtained from *AssessCY* Task is Pap II-p,g or Pap IIID-1 and the patient Age collected from the *SelectPatient* Task of the *Identification* Stage is less than 30.

```

1 Stage Cardiogram
2   label = 'Cardiogram'
3   // ... Stage elements
4
5 Stage MRIScan
6   label = 'MRI Scan'
7   // ... Stage elements
8
9 Stage Assessment
10  label = "Assessment"
11
12 Precondition
13   previousStep = 'Cardiogram'
14   previousStep = 'MRIScan'
15   // ... Other Stage elements

```

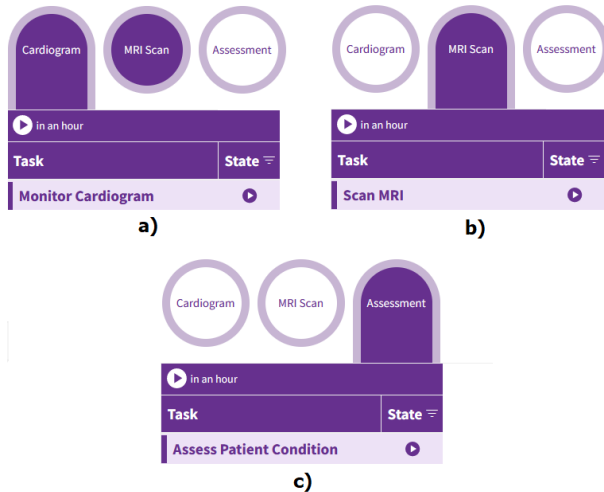


Figure 5.11: Acadela code (left) to declaring conjunction (AND) relationship between prerequisite Stages or Tasks in Acadela. The SACM UIs (right) display the *Assessment* Stage is activated only when the *Cardiogram* and *MRI Scan* Stages finish (Picture c). If both or one of the *Cardiogram* or *MRI Scan* Stages are activated (Picture a and b), the *Assessment* Stage is disabled.

5.3.7 Trigger (HttpHook)

SACM updates medical status in external systems by sending a HTTP request (HttpHook) with the data from a Case, Stage, or Task when a state change *event* (e.g., *activated*, *completed*) occurs (Michel, 2020, p. 113). Specifically, a Case supports sending an external request in each *activate*, *complete*, *terminate*, and *delete* event. Meanwhile, a Stage or Task can send multiple requests to different external systems for each state change event. Therefore, Acadela defines a Trigger element to group all HttpHooks definitions in a Case, Stage, or Task.

The CP element determines the scope of its HttpHook. Specifically, a Task HttpHook sends the Task data from its InputFields and OutputFields to the external system *url* using a *HTTP method*. Similarly, a Stage HttpHook sends all data from its Tasks, and a Case HttpHook sends the data of all the Stages and Tasks. If the request fails, SACM shows the value of the *failureMessage* attribute to the user. Table 5.8 summarizes the properties of an HttpHook definition. The position of *method*, *url*, and *failureMessage* attributes are interchangeable in Acadela. Listing 5.11 shows how Acadela defines a Trigger construct inside the *MeasureBMI* Task. Listing 5.12 presents the body content of a HTTP Request sent after completing the (MeasureBMI) Task in Listing 5.11.

SACM Attribute	Acadela Expression	Description
<code>on*</code>	On <event>	Declare the state change event that sends a HTTP request to an external service. The event name is case-insensitive in Acadela, i.e., UPPERCASE, lowercase, or camelCase are accepted. Possible Stage and Task events are available, enable, activate, complete, terminate, and correct. For DualTask, the supported events are <i>activateHumanPart</i> , <i>activateAutomatedPart</i> , <i>completeHumanPart</i> , <i>completeAutomatedPart</i> , <i>correctHumanPart</i> , and <i>correctAutomatedPart</i> . Eligible Case events are activate, complete, terminate, and delete.

5 Language Design

<code>url*</code>	invoke '<url>'	Declare the URL of an external service or system API that receives the HTTP Request.
<code>method*</code>	method <HttpMethod>	State the HTTP method used in the request. Supported methods are <i>POST</i> , <i>GET</i> , <i>PUT</i> , <i>DELETE</i> . The Method name is case-insensitive in Acadela. Note that a Case HttpHook does not have the <i>method</i> attribute.
<code>failureMessage</code>	with failureMessage '<error message>'	Declare a human readable error message in case the HTTP Request execution fails. SACM displays the <i>failureMessage</i> as an user-friendly Alert. Note that a Case HttpHook does not have the <i>failureMessage</i> attribute.

Table 5.8: Description on the attributes of a HttpHook (Michel, 2020, p. 113).

* - the attribute is required.

```

1 HumanTask MeasureBmi externalId = 'measureBmiOperation'
2   label = 'MeasureBmi'
3
4 Trigger
5   On activate method Post invoke 'https://extSystem.com/api/bmi'
6
7   On complete method Post invoke 'https://extSystem.com/api/bmi'
8   with failureMessage 'Cannot store BMI data in the partner system!'
9
10 HumanTask MeasureBmi externalId = 'measureBmiOperation'
11   label = 'Measure BMI'
12
13 Form BMIForm
14   InputField Height #number(0-3) #exactlyOne
15     label = 'Height (m):'
16
17   InputField Weight #number(0-300) #exactlyOne
18     label = 'Weight (kg):'
19
20   OutputField BmiScore #number
21     externalId = "BmiValueExternal"
22     label = 'BMI Calculation:'
23     expression = 'round(Weight / (Height * Height))'

```

Listing 5.11: Declaration of a HttpHook in a HumanTask.

```

1 {
2   "isOverdue": false,
3   "stateTransitions": {
4     // Other State activities Info
5     "COMPLETED": {
6       "by": {
7         "id": "2c9480845bee03e7015bfcad28990010",
8         "email": "practitioner.email@clinic.de",
9         "name": "Doctor James",
10        "resourceType": "users"
11      },
12      "date": "2022-11-04 19:01:25.0"
13    }
14  },
15  "parentStage": "1xfdrfg925rl6", // Evaluation Stage ID in SACM
16  "state": "COMPLETED",

```

```

17 "externalId": "measureBmiOperation",
18 "id": "uja8x6ouh6ci", // Measure BMI Task ID in SACM
19 "description": "Measure BMI",
20 "name": "ST1_MeasureBmi",
21 "resourceType": "humantasks",
22 // ... Other attributes
23 "taskParams": [
24   {
25     "task": "uja8x6ouh6ci",
26     "multiplicity": "exactlyOne",
27     "attributeType": "number",
28     "id": "vkdgza9trevj", // Height InputField ID in SACM
29     "values": [ 1.8 ],
30     "description": "Height (m):",
31     "isMandatory": true,
32     "name": "Height",
33     "isReadOnly": false,
34   },
35   {
36     "task": "uja8x6ouh6ci",
37     "multiplicity": "exactlyOne",
38     "attributeType": "number",
39     "id": "1nybzswudgv2", // Weight InputField ID in SACM
40     "values": [ 83 ],
41     "description": "Weight (kg):",
42     "isMandatory": true,
43     "name": "Weight",
44     "isReadOnly": false,
45   },
46   {
47     "task": "uja8x6ouh6ci",
48     "attributeType": "number",
49     "externalId": "BmiValueExternal",
50     "id": "s11p2i7w4jx3", // BmiScore OutputField ID in SACM
51     "values": [ 26 ],
52     "description": "BMI Calculation:",
53     "isMandatory": true,
54     "name": "BmiScore",
55     "isReadOnly": true,
56   }
57 ],
58 "isMandatory": true,
59 "case": "1oteh56xcg5s1" // Patient Case ID
60 // ... Other Task Attributes
61 }

```

Listing 5.12: Excerpt of the HTTP Request body sent by the second HttpHook in Listing 5.11 after completing the MeasureBMI Task. Appendix A.6 lists the full body content.

5.3.8 Task

A Task represents the activity of collecting medical data relevant to the health status of the patient (Michel, 2020, p. 79). Therefore, A Task can contain *questionnaires*, *data fields*, and supplemental *medical documents* to assist care professionals in conducting their activities. To model Tasks, SACM supports Human-Task, AutomatedTask, and DualTask concepts to define manual, automatic, and hybrid activities. Table 5.9 describes the attributes of a Task.

5 Language Design

SACM Attribute	Acadela Expression	Description
<code>id *</code>	<code><ID></code>	The identifier used as a reference within the Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is "name" as textX uses the "name" property as a reference to identify an imported element in a file.
<code>description *</code>	<code>label = "<text>"</code>	A text displaying the Task name in the UI. Acadela replaces the <i>description</i> keyword with the <i>label</i> term because the attribute does not explain the purpose of the Task. This description can be expressed using the local human language.
<code>multiplicity</code>	<code>#atLeastOne</code> <code>#exactlyOne (default)</code> <code>#maximalOne</code> <code>#any</code>	Declares the expected quantity of value in the InputField. For example, a multiple-answer multiple-choice question can have <code>#atLeastOne</code> or <code>#any</code> <i>multiplicity</i> as more than one answer can be added to the InputField value. Available <i>multiplicity</i> options are <code>#atLeastOne</code> , <code>#exactlyOne</code> , <code>#maximalOne</code> , or <code>#any</code> .
<code>additionalDescription</code>	<code>additionalDescription = "<text>"</code>	a text expresses explanatory information related to the current InputField. When declaring the <i>additionalDescription</i> value, SACM creates a question mark icon next to the InputField. When hovering over this icon, the UI displays the <i>additionalDescription</i> as a tooltip text.
<code>externalId</code>	<code>externalId = "<text>"</code>	Declares an ID to map with an external system
<code>ownerPath</code>	<code>owner = "<path>"</code>	The path refers to a medical group or professional responsible for executing the Task. This path begins from the Case root Entity, with every path section separated by the dot. The <i>ownerPath</i> typically starts from the Setting element, e.g., <i>Setting.Clinicians</i> for assigning members in the clinician group to handle the Task. Note that if an <i>ownerPath</i> is not declared in a Task, Acadela automatically sets the Stage <i>ownerPath</i> as the Task <i>owner</i>
<code>dueDatePath</code>	<code>dueDate = "<path>"</code>	The path refers to an Attribute stating the default deadline time to complete the Task, e.g., 24 hours from now. This path begins from the Case root Entity, with every path section separated by the dot. The <i>dueDatePath</i> typically starts from the Setting element, e.g., <i>Setting.DueDate24Hours</i> . See section 5.3.12 for the declaration of the <i>due date</i> attribute in the Case Setting. The dueDatePath is not applicable to Automated-Task

5 Language Design

isMandatory*	#mandatory (default) #notMandatory	States whether the Task must contain a value or not to complete a Task. See Section 5.3.2 for further information.
repeatable	#noRepeat (default) #repeatSerial #repeatParallel	States whether the Task is executed once (#noRepeat), serially repetitive or parallelly repetitive
activation	#manualActivate #autoActivate #activateWhen(condition) ^A	<p>States how to trigger the Task. Acadela expresses the three activation modes of SACM as follows:</p> <p>#manualActivate: triggered by a staff</p> <p>#autoActivate: triggered by the SACM execution engine</p> <p>#activateWhen(condition): triggered by the SACM execution engine when a given <i>condition</i> is satisfied.</p> <p>The #activateWhen(condition) is an Acadela construct that combines the <i>manualActivationExpression</i> and <i>activation="EXPRESSION"</i> declarations when defining a conditional Task activation.</p>
entityAttachPath	entityAttachPath = "<path>"	Declares the path to store the newly instantiated Task Entity. This path starts from the Case root Entity, hence the first section of the path is the parent Stage containing the Task. As a result, Acadela sets the default <i>entityAttachPath</i> as <parentStageID>.<currentTaskId> . Modelers can overwrite this path by declaring their <i>entityAttachPath</i> in the Task.
entityDefinitionId	entityDefinitionId = "<path>"	States the EntityDefinition ID that serves as a schema for the Task. SACM will instantiate the Entity and attach it to the declared <i>entityAttachPath</i> . By default, Acadela creates an EntityDefinition of the Task from its attributes, InputField, and OutputField elements. This EntityDefinition has the same <i>ID</i> and <i>description</i> as the Task. Therefore, Acadela sets the default <i>entityDefinitionId</i> as the Task ID . Modelers can overwrite this path by declaring their <i>entityDefinitionId</i> in the Task.
dynamicDescriptionPath	dynamicDescriptionPath = "<path>"	Declares the path to any CP element (e.g., OutputField, InputField, Task) for extending the current Task definition. This path begins from the Case root Entity, thus the first path section is a Stage ID if the element is a workflow item, i.e., Task, InputField, or OutputField; or "Setting" if the element is a Case Setting's Attribute.

Table 5.9: The attributes of an Acadela Task.

* - the attribute is required.

A - the attribute is created in Acadela and does not exist in SACM.

5 Language Design

SACM supports manual activation of (parallelly) *repeatable* Tasks. Figure 5.12 shows the Acadela code and SACM visualization of a single Task and *manually repeatable* Tasks. However, manually creating repeatable Tasks can be time-consuming to care professionals, hence Acadela supports declaring the number of *parallel instances* of the Task using `#repeatParallel(< number of parallel Task instances>)`. Note that the e-Health system, not Acadela, offers the feature to construct multiple simultaneous Task instances. SACM does not support automatic instantiations of parallelly repeatable Tasks.

```

1 Stage LabTest
2   label = 'Lab Test'
3
4   Precondition
5     previousStep = 'Identification'
6
7   HumanTask QuickTest
8     label = 'Quick Test'
9     // QuickTest Form declaration
10
11  HumanTask PcrTest
12    #manualActivate #repeatParallel
13    label = 'PCR Test'
14    dueDateRef
15      = 'Setting.WorkplanDueDate'
16
17    // PcrTest Form declaration

```

Figure 5.12: Acadela Declaration (left) and SACM UI (right) of a single Task (Quick Test) and multiple instances of a repeatable Task (PCR Test) for the *Lab Test* Stage. An **Add Task** button exists when there is at least one manually activated Task in the Stage. Clicking the **Add Task** button allows the creation of the repeatable Task(s), as shown in the below dropdown box. The example inspires by the research on applying parallel PCR Tests to detect SARS-CoV-2 by (Perchetti et al., 2020, p. 2)

Regarding each Task type, HumanTasks denote activities conducted by *staff*, possibly with a declared *due date*. AutomatedTasks are executions performed by *external systems* which does not support a *due date* declaration (Michel, 2020, p. 110). Finally, DualTasks include activities performed by *human staff*, followed by an execution conducted by an *external system* (Michel, 2020, p. 108). As a result, each InputField or OutputField of a DualTask has a duty directive to state whether a *human* or *system* executes the Task. The *due date* is declarable in DualTasks.

In Acadela syntax, each Task type has a Form containing InputFields to record medical data and OutputField to express or visualize medical information based on the InputFields. Moreover, a Task contains Precondition(s) to define *activation conditions* and enable *repetition*. Furthermore, multiple HttpHooks is includible to support activity synchronization with external systems, particularly for DualTasks. Table 5.9 lists all attributes of a Task in Acadela. Listing 5.13, and 5.14 demonstrates a complete definition of HumanTask and DualTask in Acadela. AutomatedTask definition is identical to HumanTask except that the *due date* is not declarable.

```

1 // Minimum HumanTask Declartion
2 HumanTask TASKNAME
3   label = 'TASK_LABEL'
4
5 // Complete Human Task Declaration
6 HumanTask TASKNAME
7   #mandatory // isMandatory true or false

```

5 Language Design

```
8 #noRepeat // one-time, serial or parallel repetition
9 #autoActivate // activation mode
10 #any // multiplicity
11 owner = 'Setting.<GroupOrUser>'
12 label = 'TASK_LABEL'
13 dueDateRef = 'Setting.DATE_ATTRIBUTE_ID'
14 additionalDescription = 'TOOLTIP_TEXT'
15 externalId = 'EXTERNAL_ID'
16 dynamicDescriptionRef = 'PATH_TO_OTHER_CP_OBJECT'
17
18 Precondition previousStep = 'PREVIOUS_STAGE'
19   condition = 'TRANSITION_CONDITION'
20
21 Trigger
22   On TASK_LIFECYCLE_STATE invoke 'EXTERNAL_SYSTEM_URL' method
23     HTTP_METHOD
24     with failureMessage 'ERROR_MESSAGE'
25
26   On TASK_LIFECYCLE_STATE invoke 'EXTERNAL_SYSTEM_URL' method
27     HTTP_METHOD
28     with failureMessage 'ERROR_MESSAGE'
29
30 Form FORMNAME
31   InputField INPUT_FIELD_NAME1 #text
32     label = 'FIELD_LABEL1'
33
34   OutputField FIELDNAME
35     label = 'FIELD LABEL'
36     expression = 'if (FIELD_NAME < NUM) then "OUTPUT1" else "OUTPUT2"'
```

Listing 5.13: Example of a minimum (lines 2-3) and complete HumanTask declaration (lines 6-34) in Acadela. The *directives* and *attributes* appear in the complete HumanTask definition but not in the minimum version are default or optional values that do not need to be declared.

```
1 DualTask TASKNAME
2   label = 'TASK_LABEL'
3
4 DualTask DUALTASKNAME
5   #mandatory // isMandatory true or false
6   #noRepeat // one-time, serial or parallel repetition
7   #autoActivate // activation mode
8   #any // multiplicity
9
10  owner = 'Setting.<GroupOrUser>'
11  label = 'TASK_LABEL'
12  dueDateRef = 'Setting.DATE_ATTRIBUTE_ID'
13  additionalDescription = 'TOOLTIP_TEXT'
14  externalId = 'EXTERNAL_ID'
15  dynamicDescriptionRef = 'PATH_TO_OTHER_CP_OBJECT'
16
17  Precondition previousStep = 'PREREQUISITE_STAGE_OR_TASK_ID'
18    condition = 'TRANSITION_CONDITION'
19
20  Trigger
21    On TASK_LIFECYCLE_STATE
22      invoke 'EXTERNAL_SYSTEM_URL' method HTTP_METHOD
23      with failureMessage 'ERROR_MESSAGE'
24
25
26  Form FORMNAME
```


5 Language Design

```

27     InputField INPUT_FIELD_NAME1 #text #humanDuty
28         label = 'FIELD_LABEL1'
29
30     InputField INPUT_FIELD_NAME2 #json #systemDuty
31         uiRef = 'linediagram'
32         label = 'FIELD_LABEL2'
33
34     OutputField FIELDNAME #left
35         label = 'OUTPUT FIELD LABEL'
36         expression = 'if (FIELD_NAME < NUM and FIELD_NAME >= NUM)
37             then "OUTPUT1"
38             else if (FIELD_NAME = NUM or FIELD_NAME = "TEXT")
39                 then "OUTPUT2"
40             else "OUTPUT3"'

```

Listing 5.14: Example of a minimum (lines 1-2) and complete DualTask declaration (lines 4-40) in Acadela.

5.3.9 Stage

A Stage represents a treatment phase comprising multiple Tasks. Similar to a Task, a Stage has an *owner* which is a group, or an individual that executes the Stage operations and reads or writes Stage data. SACM supports the definition of *one-time*, *parallelly*, or *serially repeatable* Stages. Table 5.10 describes the Stage attributes and their expression in Acadela.

SACM Attribute	Acadela Expression	Description
<code>id *</code>	<code><ID></code>	The identifier used as a reference within the Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is " <i>name</i> " as <code>textX</code> uses the " <i>name</i> " property as a reference to identify an imported element in a file.
<code>description *</code>	<code>label = "<text>"</code>	A text displaying the Stage name in the UI. Acadela replaces the <i>description</i> keyword with the <i>label</i> term because SACM renders the <i>description</i> Attribute as the Stage name, not as an explanation of the Stage purpose. The <i>description</i> value can be expressed using the local human language.
<code>multiplicity</code>	<code>#atLeastOne</code> <code>#exactlyOne(default)</code> <code>#maximalOne</code> <code>#any</code>	Declares the expected multitude of a Stage. Serially and parallelly repeatable Stages shall have multiplicity of <code>#atLeastOne</code> . Declaring <code>#any</code> deactivates conditional trigger of a Stage.
<code>additionalDescription</code>	<code>additionalDescription = "<text>"</code>	a text expresses explanatory information related to the current InputField. When declaring the <i>additionalDescription</i> value, SACM creates a question mark icon next to the InputField. When hovering over this icon, the UI displays the <i>additionalDescription</i> as a tooltip text.
<code>externalId</code>	<code>externalId = "<text>"</code>	Declares an ID to map with an external system

5 Language Design

<code>ownerPath</code>	<code>owner = "<path>"</code>	The path refers to a medical group or professional responsible for executing the Task. This path begins from the Case root Entity, with every path section separated by the dot. The <i>ownerPath</i> typically starts from the Setting element, e.g., <i>Setting.Clinicians</i> for assigning members in the clinician group to handle the Task.
<code>dueDatePath</code>	<code>dueDate = "<path>"</code>	The path refers to an Attribute stating the default deadline time to complete the Stage, e.g., 24 hours from now. This path begins from the Case root Entity, with every path section separated by the dot. The <i>dueDatePath</i> typically starts from the Setting element, e.g., <i>Setting.DueDate24Hours</i> . See section 5.3.12 for the declaration of the due date attribute in the Case Setting.
<code>isMandatory*</code>	<code>#mandatory</code> (default) <code>#notMandatory</code>	States whether the Stage must be completed once activated to accomplish a Case treatment. See Section 5.3.2 for further information.
<code>repeatable</code>	<code>#noRepeat</code> (default) <code>#repeatSerial</code> <code>#repeatParallel</code>	States whether the Stage is executed once (<code>#noRepeat</code>), serially repetitive (<code>#repeatSerial</code>) or parallelly repetitive (<code>#repeatParallel</code>)
<code>activation</code>	<code>#manualActivate</code> <code>#autoActivate</code> <code>#activateWhen(condition)</code> ^A	States how to trigger the Stage. Acadela expresses the three activation modes as follows: <code>#manualActivate</code> : triggered by a human staff <code>#autoActivate</code> : triggered by the SACM execution engine <code>#activateWhen(condition)</code> : triggered by the SACM execution engine when a given condition is satisfied. The <code>#activateWhen(condition)</code> is an Acadela construct that combines the <i>manualActivationExpression="<conditionalExpression>"</i> and <i>activation="EXPRESSION"</i> declarations when defining a conditional Stage activation in SACM.
<code>entityAttachPath</code>	<code>entityAttachPath = "<path>"</code>	Declares the path to store the newly instantiated Stage Entity. This path starts from the Case root Entity, hence the default path value is the Stage ID . Modelers can overwrite this path by declaring their <i>entityAttachPath</i> attribute.
<code>dynamicDescriptionPath</code>	<code>dynamicDescriptionPath='<path>'</code>	Declares the path to a CP element (e.g., InputField, Task) for extending the current Stage definition. This path begins from the Case root Entity, thus the first path section is a Stage ID if the element is a workflow item, i.e., Stage, Task, InputField, or OutputField; or "Setting" if the element is a Setting Attribute.

entityDefinitionId	entityDefinitionId=' <path>'	States the EntityDefinition ID that serves as a schema for the Stage. SACM will instantiate the Entity and attach it to the declared <i>entityAttachPath</i> . By default, Acadela creates an EntityDefinition of the Stage from its attributes and Tasks. This EntityDefinition has the same ID and <i>description</i> as the Stage. Therefore, the default <i>entityDefinitionId</i> is the Stage ID in Acadela. Modelers can overwrite this path by declaring their <i>entityDefinitionId</i> attribute.
--------------------	------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 5.10: The attributes of an Acadela Stage.

! - the attribute is required.

A - the attribute is created in Acadela and does not exist in SACM.

Like *parallel* Tasks, SACM provides a *manual instantiation* for *parallelly repetitive* Stages but does not support an automatic mechanism to create the Stages. Each parallel Stage is executable only once. Figure 5.13 shows the Acadela definition (left) and UI visualization (right) of a single Stage and manually activated, parallelly repetitive Stages. While Listing 5.15 and Figure 5.14 illustrate the definition and visualization of a serially repeatable Stage. Nested Stages are feasible but not implemented in SACM.

Regarding the structure, a Stage can have multiple Tasks to record data or assist care professionals with visualization of medical information. Moreover, Similar to Tasks, a Stage contains Precondition(s) to control the process flow. Furthermore, HttpHooks are includible to share all the Tasks data in the Stage with external systems. Listing 5.16 demonstrates a minimum and complete definition of attributes and child elements in a Stage.

```

1 // Single Stage
2 Stage Identification
3   label = 'Identification'
4   // ... Stage Elements
5
6 // Parallelly repeatable Stage
7 Stage PcrTest
8   #manualActivate #repeatParallel
9   label = 'PCR Test'
10  // ... Stage Elements

```

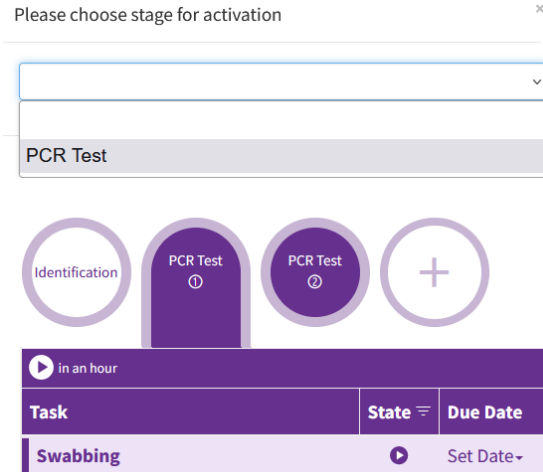


Figure 5.13: Acadela Declaration (left) and SACM UI (right) of a single Stage (Identification) and parallelly repeatable Stage (PCR Test). The circle with a plus icon exists when declaring at least one manually activable Stage. Clicking the plus circle displays a dropdown box to select the (repeatable) Stage, which SACM will instantiate. The number below the parallel Stage is the cardinal instance number, not the latest iteration of that Stage. Each parallel Stage is executable only once. The example inspires by the research on applying parallel PCR Tests to detect SARS-CoV-2 by (Perchetti et al., 2020, p. 2)

```

1 Stage Identification
2   label = "Identification"
3   // ... Stage elements
4
5 Stage Evaluation
6   #repeatSerial #atLeastOne
7   label = "Exercise"
8
9   Precondition
10    previousStep = 'Identification'
11
12   Precondition
13    previousStep = 'Evaluation'

```

Listing 5.15: Precondition Definition to enforce a repetitive Stage using the previousStep Attribute. Lines 9 and 10 state that the Exercise Stage is only triggered when the Identification Stage is completed. Lines 11 and 12 state that the Exercise Stage is repetitive as the previous Exercise Stage iteration needs to be finished to activate the new Exercise Stage

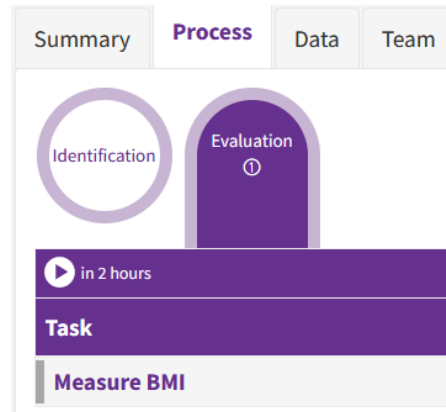


Figure 5.14: UI display of a repetitive Stage in SACM. A blank circle expresses a completed Stage. The door-shaped purple object denotes the activated *Exercise* Stage that the user is interacting with. SACM shows the latest Stage iteration below its name.

```

1 // Minimum declaration of a single execution stage
2 Stage STAGENAME
3   label = 'STAGE_LABEL'
4   // Task(s) Definition
5
6 // Complete Stage Declaration
7 Stage STAGENAME
8   #mandatory
9   #noRepeat
10  #autoActivate
11  #any
12  owner = 'Setting.CaseOwner'
13  label = 'STAGE_LABEL'
14  additionalDescription = 'ADDITIONAL_DESCRIPTION'
15  externalId = 'EXTERNAL_ID'
16  dynamicDescriptionRef = 'PATH_TO_OBJ_OF_DYNAMIC_DESCRIPTION'
17
18 // Precondition(s) Definition
19 // HttpHook(s) Definition
20 // Task(s) Definition

```

Listing 5.16: Minimum (lines 2-4) and complete declaration (lines 6-19) of a Stage in Acalada

5.3.10 Summary Panel

A SummaryPanel contains critical InputFields or OutputFields values in a readonly format to present the *patient medical status* or *treatment goal(s)*. The SummaryPanel includes each InputField or OutputField value using a *reference path* starting from its parent Stage, i.e., <StageId>.<TaskId>.<Input/OutputFieldId>.

SACM applies a grid layout with three columns format to display the SummaryPanel's values. The visual effect of the value (e.g., background color, SVG rendering) remains in the SummaryPanel.

SACM Attribute	Acadela Expression	Description
<code>id *</code>	<code><ID></code>	The identifier used as a reference within the Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is " <i>name</i> " as textX uses the " <i>name</i> " property as a reference to identify an imported element in a file.
<code>description *</code>	<code>label = "<text>"</code>	The Summary title in the UI. Acadela replaces the <i>description</i> keyword with the <i>label</i> term because SACM renders the <i>description</i> Attribute as the Summary title, not as an explanation of the Summary value. The <i>description</i> value can be expressed using the local human language.
<code>position</code>	<code>#left</code> <code>#center</code> <code>#right</code> <code>#stretched</code>	States the grid layout position to render the InputField in the UI, which contains three grid columns. The supported values are: <code>#left</code> : the left grid cell <code>#center</code> : the middle grid cell <code>#right</code> : the right grid cell <code>#stretched</code> : span all three cells

Table 5.11: The attributes of an Acadela SummaryPanel.

* - the attribute is required.

```

1 SummaryPanel
2   Section SECTION_NAME_1
3     label = "Section 1 Title"
4     InfoPath <StageID>.<TaskID>.<InputField1ID>
5
6   Section SECTION_NAME_2
7     #left // column position in the Grid Layout
8     label = "Section 2 Title"
9     InfoPath <StageID>.<TaskID>.<InputField2ID>
10    InfoPath <StageID>.<TaskID>.<InputField3ID>
11    InfoPath <StageID>.<TaskID>.<OutputField1ID>

```

Listing 5.17: SummaryPanel definition in Acadela syntax. A SummaryPanel can contain multiple Sections, each Section has a position directive, a title, and one or many InfoPath which points to an Input/OutputField in the Case Definition from its parent Stage or Task.

```

1 SummaryPanel
2   Section TreatmentGoal #left
3     label = "Treatment Goal:"
4     InfoPath TreatmentApproval.DiscussTreatment.TreatmentGoal
5
6   Section SleepCondition #left
7     label = "Sleep Condition"
8     InfoPath Questioning.QuestionPatientCondition.SleepCondition
9
10  Section MessageConsent #left
11    label = "Remedies Consent:"
12    InfoPath TreatmentApproval.DiscussTreatment.MessageConsent
13    InfoPath TreatmentApproval.DiscussTreatment.AcupunctureConsent
14    InfoPath TreatmentApproval.DiscussTreatment.GuashaConsent
15
16  Section MassagePosition #center

```

5 Language Design

```

17     label = "Massaging Points"
18     InfoPath Messaging.MessageHead.MessageLocationVisual
19
20 Stage Questioning // Stage Attributes
21
22 HumanTask QuestionPatientCondition // Task Attributes
23
24 Form PatientConditionForm
25     InputField SleepCondition #singleChoice
26     question = "How is your sleep condition?"
27     option "I got nightmare frequently" value = '1'
28     // Other options
29     option "Good" value = '5'
30     option "Very good" value = '6'
31
32 Stage TreatmentApproval // Stage Attributes
33
34 HumanTask DiscussTreatment // Task Attributes
35
36 Form TreatmentDiscussionForm
37     InputField TreatmentGoal #text
38     label = "Treatment Goal:"
39
40     InputField MessageConsent #singleChoice
41     question = "Message Consent:"
42     option "No" value = '0'
43     option "Yes" value = '1'
44     uiRef = "colors(0 <= red < 1 <= green < 2)"

```

Listing 5.18: Example of a SummaryPanel Definition to display *treatment goal* and *medical data* in the CP at a specific grid layout position. Listing A.9 shows the dynamic image rendering mechanism used by the *MessageLocationVisual* OutputField.

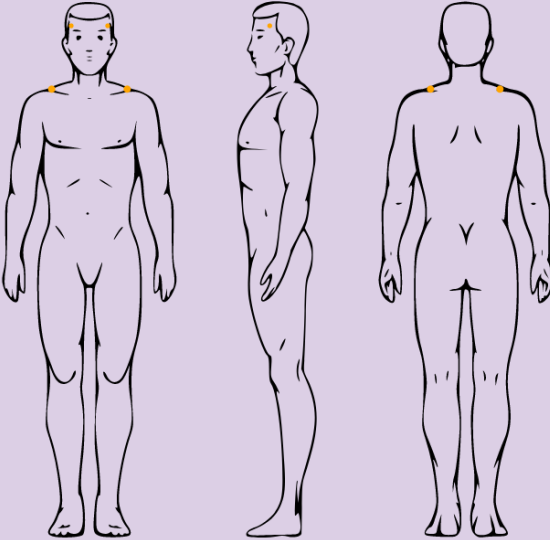
Treatment Goal:	Massaging Points	Remedies Consent:
Treatment Goal: Relieve Periodic Pain		Message Consent: Yes
Cause How did it start? After overwork with computer		Acupuncture Consent: Yes
Frequency How often is the headache? Every several days		Guasha Consent: No
Sleep Condition How is your sleep condition? Often sleep well		

Figure 5.15: Illustration of SummaryPanel UI in SACM as defined in Listing 5.18. SACM retains the visual effect of each Input/OutputField.

5.3.11 Responsibilities

SACM provides a separate API to create Users and Groups representing medical professionals and teams. Modelers define the information of Users and Groups according to the SACM XML structure. Specifically, a SACM User definition contains a *local ID*, *static ID* to assist referencing from external systems, *name*, *role*, *age*, and other organizational data. Meanwhile, a SACM Group definition contains a *local ID*, **static ID**, **name**, and **title** that appears in the UI. Table 5.12 and 5.13 list the attributes in Acadela User and Group elements.

SACM Attribute	Acadela Expression	Description
<code>id *</code>	<code><ID></code>	The identifier used as a reference within the Acadela Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is " <i>name</i> " as textX uses the " <i>name</i> " property as a reference to identify an imported element in a file.
<code>staticId</code>	<code>staticId = "<text>"</code>	The unique identifier of this User in SACM, which external systems use as a reference. In SACM, the maximum length of a <i>staticId</i> is 32 characters.

Table 5.12: The attributes of an Acadela User.

* - the attribute is required.

SACM Attribute	Acadela Expression	Description
<code>id *</code>	<code><ID></code>	The identifier used as a reference within the Acadela Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is " <i>name</i> " as textX uses the " <i>name</i> " property as a reference to identify an imported element in a file.
<code>staticId</code>	<code>staticId = "<text>"</code>	The unique identifier of this Group in SACM, which external systems use as a reference. In SACM, the maximum length of a <i>staticId</i> is 32 characters.

Table 5.13: The attributes of an Acadela Group.

* - the attribute is required.

Acadela provides the Group and User constructs to declare references to SACM Groups and Users. Each Acadela Group or User has a 1) SACM local ID that is referable within a CP and 2) an optional *staticId* attribute that stores the global identifier of the medical team or staff in SACM. In addition, Group elements have a *name* attribute storing the medical team name. This name attribute helps Acadela verify the existence of a declared medical team in the CP. Note that SACM requires the *staticId* in the declaration of each Group or User, yet to enhance usability, Acadela automatically searches the *staticId* from the SACM database. If the *staticId* is declared, then Acadela skips this search operation. Tables 5.12 and 5.13 list the attributes of the User and Group elements in Acadela.

Acadela clusters the Group and User definitions under a Responsibilities element. SACM requires the Group or User declarations as Attributes in the Case Setting, hence Stages and Tasks can refer to them as the process *owner* when applicable. Listing 5.19 and 5.20 show the concrete syntax and example of defining references to medical teams and professionals in the Responsibilities element. Section Setting describe the declaration of the Group and User aliases in Listing 5.21.

```

1 Responsibilities
2   Group <id> name = '<groupName>' staticId = '<GroupStaticIdInSacm>'
3   User <id> staticId = '<UserStaticIdInSacm>'

```

Listing 5.19: Group and User Definition in Acadela syntax.


```

1 Responsibilities
2   group DemoClinicians name = 'Demo Clinician'
3     staticId = "5f737af3443311e9bd8a0242ac13000e"
4   group DemoProfessionals name = 'Demo Professional' // no staticId,
      Acadela will search from the name
5   user alanF staticId = "349ca79c443011e9bd8a0242ac13000e"
6   user maryL // no staticId, Acadela will search from the user ID (maryL)

```

Listing 5.20: Group and User Definition in Acadela syntax.

5.3.12 Setting

The Setting is the central element to store the *case owner*, *due date*, and *aliases* of Groups or Users as Attributes. Optionally, modelers can declare a *case client* to enable searching a patient from the patient list in SACM. Table 5.14 shows the properties of a Setting Attribute declared in the Setting element.

SACM Attribute	Acadela Expression	Description
id *	<ID>	The identifier used as a reference within the Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is "name" as textX uses the "name" property as a reference to identify an imported element in a file.
description *	label = "<text>"	A text displayed in the UI. Acadela replaces the <i>description</i> keyword with the <i>label</i> term to emphasize the UI rendering characteristic of this attribute. This description can be expressed using the local human language.
type *	#<type> (See Section 5.3.1)	Enforce a data type constraint (e.g., text, number, multiple-choice question, link to a Case object) on the Attribute. See Section 5.3.1 for a detailed description of type in Acadela.
multiplicity	#atLeastOne #exactOne(default) #maximalOne #any	Declares the expected number of Attribute instances at runtime.
additionalDescription	additionalDescription = "<text>"	a text expresses explanatory information related to the current Attribute. When declaring the <i>additionalDescription</i> value.
uiReference	uiRef = "<text>"	Express the UI effect applied to this Attribute. For example, define a background color band for numeric value, or display the Attribute as a URL link to an external file.
externalId	externalId = "<text>"	Declares an ID to map with an external system
defaultValue	defaultValue = "<text>"	Declares a single value that SACM will initially set to the Attribute
defaultValues	defaultValues = '<text>'	Declares a list of multiple values that SACM will initially set to the Attribute

Table 5.14: The attributes of an Acadela Setting Attribute.

* - the attribute is required.

5 Language Design

The Setting is also suitable to store *constants* or *global variables*. InputFields and OutputField can leverage the *global variables* to store, compute, or display the data, which are required by multiple Tasks or Stages. Acadela supports the read (for OutputFields) or write (for InputFields) of *global variables* with a `#custom` directive and ElementPath attribute, e.g., ElementPath="Setting.<globalVariableAttributeId>" (See the *path* attribute of Section 5.3.3 and 5.3.4). Listing 5.22 shows an example definition of a Case Setting in Acadela.

```
1 Responsibilities
2   group DemoClinicians name = 'Demo Clinician' staticId = "5
3     f737af3443311e9bd8a0242ac13000e"
4     // ... Other Group and User references
5 Setting
6   CaseOwner DemoClinicians
7     label = 'Demo Clinicians'
8
9   CasePatient DemoPatients
10    label = 'Patient'
11
12  Attribute Clinician
13    #Link.#Users(DemoClinicians)
14    label = 'Clinician'
15
16  Attribute WorkplanDueDate
17    #date.#after(TODAY) // due date within the next 24 hours
18    label = '24-hour Due Date'
```

Listing 5.21: Case Setting declaration syntax in Acadela.

5.3.13 Case

An Acadela Case represents a CP model that comprises the Case attributes listed in Table 5.13 and the following elements from SACM CaseDefinition (Michel, 2020, p. 103):

1. **Responsibilities:** The reference to medical teams or professionals involved in the CP.
2. **Case Setting:** Stores Case meta-data, including the *case owner*, optional *case patient*, *alias* of a Responsibilities element, *due date* duration (e.g., next 24 hours), *constants*, and *global variables*.
3. **Case HttpHook:** Declares a *HTTP request* being sent to an external service when a Case state change event occurs.
4. **Case Summary:** Presents the *critical medical data* and *treatment goal(s)* for care professionals to analyze the patient's condition and determine the necessary medical interventions.
5. **Stage(s):** The treatment *phases* consist of medical or administrative Tasks that collects or displays case-related data to support the treatment process. Preconditions are declarable to enforce control flows in Stages and Tasks, i.e., define structured processes of ordered steps or adaptive processes containing flexible alternative execution paths.

Table 5.15 describes the attributes in an Acadela Case. Listing 5.22 shows the syntax for declaring a Case in Acadela, while Listing 5.23 demonstrates the code to declare a Case and Figure 5.16 illustrates the UI of the resulting workflow model in SACM.

SACM Attribute	Acadela Expression	Description
<code>id*</code>	<code><ID></code>	The identifier used as a reference within the Acadela Case declaration or within an imported file. The attribute name of <i>ID</i> in Acadela is " <i>name</i> " as textX uses the " <i>name</i> " property as a reference to identify an imported element in a file.
<code>description*</code>	<code>label = "<text>"</code>	A text displayed in the UI to show the CP name. Acadela replaces the <i>description</i> keyword with the <i>label</i> term to emphasize the UI rendering characteristic of this attribute. This description can be expressed using the local human language.
<code>prefix *A</code>	<code>prefix = "<text>"</code>	Prepended text to the ID of the Case, Stages, and Tasks in the CP. This prefix helps SACM distinguish CPs having the same name but used by different medical institutions.
<code>version *A</code>	<code>version = <integer></code>	State the version number of the current Case definition. SACM appends this value to a prefixed Case id to support CP versioning.

Table 5.15: The attributes of an Acadela Case.

* - the attribute is required.

A - the attribute is created in Acadela and does not exist in SACM.

```

1  define Case <CpId>
2    prefix = '<prefixText>'
3    version = <versionNumber>
4    label = '<CpNameInUI>'
5
6    Trigger
7      On <caseStateEvent1> invoke '<externalSystemApiUrl1>'
8      On <caseStateEvent2> invoke '<externalSystemApiUrl2>'
9
10   // Responsibilities Definition
11   // Setting Definition
12   // Stage Definitions

```

Listing 5.22: Case declaration syntax in Acadela.

```

1  #aca0.1
2  workspace Demo
3
4  define Case Schizophrenia
5    prefix = 'MSC'
6    version = 1
7    label = 'Schizophrenia Treatment'
8
9    Trigger
10   On activate invoke 'https://partnerSystemUrl/api/activate'
11
12   Responsibilities
13   group DemoPhysicians name = 'Demo Physician' //staticId = 'asdf234'
14   group DemoClinicians name = 'Demo Clinician'
15   group DemoProfessionals name = 'Demo Professional'
16   group DemoPatients name = 'Demo Patient'
17   group DemoNurses name = 'Demo Nurse'
18

```

5 Language Design

```
19     user demoUser
20
21     Setting
22     CaseOwner DemoProfessionals #exactlyOne
23         label = 'Demo Professionals'
24
25     Attribute WorkplanDueDate
26         #exactlyOne #date.#after(TODAY)
27         label = 'Workplan Due Date'
28         externalId = 'dueDateConnie'
29
30     CasePatient DemoPatients #exactlyOne
31         label = 'Patient'
32
33     Attribute Clinician
34         #Link.#Users(DemoClinicians)
35         label = 'Clinician'
36
37     Attribute Nurse
38         #Link.#Users(DemoNurses)
39         label = 'Nurse'
40
41     SummaryPanel
42     Section MedicalInformation
43         label = "Medical Information:"
44         InfoPath Identification.MedicalInfo.Age
45         InfoPath Identification.MedicalInfo.Gender
46
47     Section PatientPreferences #left
48         label = "Patient Preferences:"
49         InfoPath Identification.PatientPreferences.TreatmentGoal
50     // Other InfoPath and Summary Sections
51
52     Stage Identification
53         label = "Identification"
54     // Stage Attributes and Task Definitions
55
56     Stage ShareDecisionMaking #repeatSerial
57         owner = 'Setting.Clinician'
58         label = 'Share Decision Making'
59
60     Precondition previousStep = 'Identification'
61     Precondition previousStep = 'ShareDecisionMaking'
62
63     HumanTask OpenTherapySession #exactlyOne
64         label = 'Arrange Therapy Session'
65     // Task Form Definition
66
67     Stage Discharge #manualActivate
68         owner = 'Setting.Nurse'
69         label = 'Discharge'
70     // Discharge Task Declaration
```

Listing 5.23: An example of Case declaration syntax in Acadela.

Figure 5.16: UI of the Case defined in Listing 5.23 from the view of a Clinician User. The screenshot shows the current workflow with completed and opening Stages (purple-background circles), Tasks, assigned roles, and other Case information (e.g., Case name, patient info).

5.3.14 Workspace

In the CONNECARE system, multiple hospitals and clinics stored their CP models in the SACM backend. Therefore, SACM defines a *Workspace* concept to represent a collection of all CPs (Case metamodels) used by a medical facility. The *Workspace id* attribute is typically a hospital or clinic ID stored in the SACM database. Besides, to support various parsing and interpreting mechanisms for CPs in different medical institutions, Acadela has an optional directive stating the e-Health system abbreviation and Acadela version number at the beginning of the file. Acadela declares this directive and the *Workspace* reference as follows:

```

1 #<eHealthSystemAbbr><versionNumber> // e.g. #sacm1.0
2 Workspace <id>
3 // Case Definition

```

Listing 5.24: Workspace Definition in Acadela syntax.

5.3.15 Import

To enhance reusability, Acadela leverages the textX Reference Resolving Expression Language (RREL) (Dejanović, n.d.c) to enable importing a Case, Setting, SummaryPanel, Stage, Task, HttpHook, Form, InputField, OutputField, Attribute, and variable into a CP definition. First, modelers define the above element(s) in a separate file with *.aca* extension. Next, the element is importable at the beginning of the Case definition, thus, modelers can use an element in the imported file by referring to their ID using the syntax `use <ElementType> <ElementId>`.

Additionally, Acadela also supports *import as aliases*, hence modelers can assign a namespace to an imported file. Listing 5.25 shows the syntax to declare and use both forms of import. Finally, Listing 5.26 shows an example of using both types of import in a Case definition.

```

1 import <pathToFolder>.<fileName1> // Import a file contains CP element(s)
2 import <pathToFolder>.<fileName2> as <alias> // import a file as alias
3
4 // CP element import
5 use <ElementType> <elementId> // call a CP element without alias
6 use <ElementType> <alias>.<elementId> // call a CP element with alias
7

```

5 Language Design

```
8 // CP element's attribute import
9 <attrKeyword> = use <attrId> // assign an attribute value without alias
10 <attrKeyword> = use <alias>.<attrId> // assign the value with alias
```

Listing 5.25: Acadela syntax for importing a CP element or an element's attribute with or without alias.

```
1 // extfile/prescribeTask.aca content:
2 define HumanTask Prescribe #repeatParallel #manualActivate
3   label = 'Prescribe'
4   // Task Form declaration
5   // -----
6
7 // extfile/bmiColorCode.aca content:
8 define bmiUiRef = 'colors(0 < yellow < 18 < green < 25 < orange < 30 <
9   red < 100)'
```

```
9
10 // -----
11
12 // Case definition
13 #aca1.0
14 import extfile.prescribeTask // containing the Prescribe HumanTask
15 import extfile.bmiColorCode as bmiColor // import as alias
16
17 workspace DemoClinic
18
19 define case MA1_Malnutrition
20   // Case attribute, Responsibilities, Setting, and HttpHook definitions
21   Stage Evaluation
22     label = "Evaluation"
23
24   Task MeasureBmi
25     label = "Measure BMI"
26     Form BmiForm
27       // Other Input Fields for Height and Weight
28       OutputField BmiScorePlus #left #number
29         label = 'BMI Score:'
30         expression = 'round(Weight / (Height * Height))'
31         uiRef = use bmiColor.bmiUiRef // call an element with alias
32
33   Stage Prescription
34     label = "Prescription"
35
36     use Task Prescribe // call an element without alias
```

Listing 5.26: Example of importing a CP element or attribute in Acadela. Suppose that the *extfile* folder stores files of a HumanTask (prescribeTask.aca) or *bmiUiRef* variable (bmiColorCode.aca) definitions. Acadela first requires import declarations before the Workspace reference (lines 14 and 15). Finally, for CP elements, modelers need to call the Task import under the Stage at the intended position using the `use Task Prescribe` structure (line 36). For alias import, modelers use the `alias.<elementID>` structure. Variable import is placed in the value part of the key=value syntax (line 31).

5.4 Constraint Validation

During the modeling process, modelers can make various syntax or semantic errors. textX already provides a syntax error handler for its parser that can catch typos and unexpected elements in the CP model. Additionally, textX also provides an error message stating the *error location* with line and row numbers, the *code snippet* around the error location with an asterisk marking the error position, and the *expected*

5 Language Design

output at that position. Although textX error messages include the name of terminals and non-terminals defined in the Acadela grammar, these names are only familiar to language designers. Thus the original error messages need to be more understandable to modelers, its primary user group.

Supporting modelers with concise and comprehensible error messages that articulate the *cause* and *position of error* with *familiar technical terms* is crucial to 1) prevent barriers in development progress (Becker et al., 2018, p. 635); 2) reduce the number of exceptions (Becker, 2016, p. 130); 3) ease debugging difficulty (Barik et al., 2017, p. 583), which results in shortening the development time, as developers typically allocate 13-25% of their development tasks for fixing errors (Barik et al., 2017, p. 575). Therefore, our first goal is to enhance textX syntax error messages using the domain concepts or terms expressed in the concrete syntax.

In addition, textX does not provide tools for semantic constraint validation. For example, Acadela should detect any non-existing path to a CP element. As a result, our study developed additional 1) *custom semantic validators* and 2) *enhanced error message generator* that reveals the error *cause*, *location*, and *fixing direction* as recommended by (Nielsen, 2001). This section describes the types of syntax and semantic errors that Acadela can detect and their enhanced output error messages. For implementation details, Section 6.5.2 explains syntax or semantic fault detection and error message generation mechanisms.

Note that the Acadela semantic and syntax error analyzers do not output all the syntax and semantic errors simultaneously, but only display the first detected fault. textX catches successive errors after modelers fix the previous one. Acadela will output a successfully compiled message if no syntax or semantic error is detected.

5.4.1 Syntax Errors

Unexpected Element

Acadela grammar defines the *hierarchical structure* and *expected position* of each CP element and its attributes. For instance, a Stage requires the definition of *directives* and *Stage attributes*, followed by *Precondition(s)*, *Trigger (HttpHooks)*, and *Task(s)* definitions. The order of the last three elements is interchangeable. Defining a CP item that does not exist at a hierarchical level or does not appear in an expected position violates this rule. For instance, declaring a Form in a Stage is invalid as Stage does not accept any Form as a children element, as shown in Figure 5.4.1; stating a *directive* after a *Precondition* infringes the expected order as directives shall be declared right after a Stage ID. Defining a non-existing modeling concept also triggers this error.

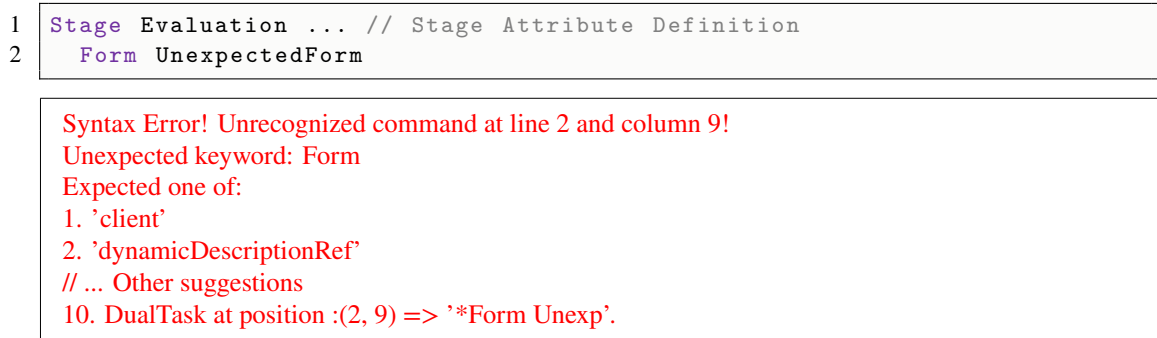


Figure 5.17: Example of invalid element declaration (top) and their enhanced error message (bottom) in Acadela. The Form *UnexpectedForm* is invalid as Stages do not accept a Form as a child element.

Another concern of textX is that it displays operators using the construct name instead of the character. For example, Acadela grammar defines the equal sign to assign a value to an attribute as follows:

```
1 Eq: // grammar construct name
2   '=' // grammar rule, an equal sign in this case
3 ; // rule terminates
```

5 Language Design

When the operator is missing, textX shows the construct name (EQ) but not the expected character (=), as shown in the end of Listing 5.4.1:

```
139 HumanTask MeasureBloodCholesterol #mandatory
140     owner 'Setting.Nurse'
141     label = 'Record Blood Cholesterol'
```

(a) Original textX error message:

```
Expected Eq at position <absolutePathToGrammarFolder>:(140, 19) => ' owner '*Setting.N'
```

(b) Acadela enhanced error message:

```
Syntax Error! Unrecognized command at line 140 and column 19!
Expected Equal sign (=) at position <absolutePathToGrammarFolder>:(140, 19) => ' owner '*Set-
ting.N'.
```

Figure 5.18: Example of the need for customizing the textX error message. Original error message (a) of textX that shows the grammar construct ('Eq') in instead of the concrete syntax element ("="). Therefore, Acadela enhances the message (b) to show the expected operator.

This action can confuse modelers if they do not know the grammar construct name of the missing element. Therefore, Acadela translates this error message to show the expected character instead.

Acadela leverages the textX parser and customizes error messages to catch four types of syntax errors.

Typo

Acadela contains a set of *keywords* to signal the declaration of CP elements (e.g., *Stage*, *Task*), *directives* (e.g., *#mandatory*, *#any*), or *attributes* (e.g., *label*, *uiRef*). By default, if modelers input a typo in the CP definition code, Acadela states that the typo word is unknown and suggests syntactically valid keywords expected at that position. However, if the typo has less than three characters different from an Acadela keyword, the error message includes a "Did you mean <keyword>?" question after stating the error type. This behavior intends to help modelers quickly realize the direction to solve the typo error. Figure 5.4.1 and 5.4.1 show the code snippet and error message to demonstrate the handling of the two typo cases.

```
97 Precnoddition
98     previousStep =
99     'Identification'
```

```
Syntax Error! Unrecognized command at line 97 and column
17!
No keyword Precnoddition. Did you mean: Precondition?
Expected one of:
1. 'externalId'
2. 'additionalDescription'
3. 'dynamicDescriptionRef'
4. Precondition
5. Trigger
6. Form
7. Ref at position :(97, 17) => '*Precnoddit'.
```

Figure 5.19: Example of an error message (right) for typos that are similar to Acadela keyword (left). At line 97, the typo "Precnoddition" has two different characters than the "Precondition" keyword of Acadela. Thus the error message suggests the correct keyword to modelers.

```

97 Prerequisite
98   previousStep =
99     'Identification'

```

```

Syntax Error! Unrecognized command at line 97 and column 17!
Unrecognized keyword: Prerequisite
Expected one of:
1. 'externalId'
2. 'additionalDescription'
3. 'dynamicDescriptionRef'
4. Precondition
5. Trigger
6. Form
7. Ref at position :(97, 17) => '*Prerequisi'.

```

Figure 5.20: Example of an error message (right) for typos that are totally different from Acadela keywords (left). In line 97, the typo "Prerequisite" does not exist in the Acadela dictionary, and many characters are not closely similar to any Acadela keyword. Thus Acadela only states that the keyword is unrecognized.

Unexpected Data Type

The value of element attributes has a specific data type. For example, the value of a CP *version* is a *number*, and of a *label* is a *string*, i.e., double-quoted or single-quoted text. textX expresses these types of data using **STRING**, **INT**, **FLOAT**, or **NUMBER** constants. However, stating the data type constant may not expose the error cause explicitly. For example, an "expected STRING" message means that the modelers forget to open or close a text with a single or double quote, as shown in Figure 5.21. Therefore, Acadela enhances the error message to reveal the root cause, thus modelers quickly realize the bug.

```

132 Stage MedicalTest #mandatory
133   label = Medical Test'

```

(a) Original textX error message:

```

Expected STRING at position <absolutePathToGrammarFolder>:(133, 17) => ' label = *Medical
Te'.

```

(b) Acadela enhanced error message:

```

Syntax Error! Unrecognized command at line 133 and column 17!
Expected Text with quotation marks ("", ") at position <absolutePathToGrammarFolder>:(133, 17)
=> ' label = *Medical Te'.

```

Figure 5.21: Example of an enhanced error message (b) in Acadela from the original one of textX (a) to explain the invalid String data type (top). In line 133, a single quote is missing in the String value. Thus Acadela informs modelers that quotation marks are needed.

Custom String Pattern

Several attributes require a String value that conforms to a unique SACM syntactic rule. For instance, the *conditional statement* of an *expression* attribute in OutputFields must respect the SACM syntax, which uses the "then" keyword to signal an action when a condition is true. Another example is the definition of a color code that must begin with a **colors** function with a parameter specifying the color bands, each consisting of a known color name (e.g., green, red) in the middle of two numbers. Acadela sets a dedicated grammar rule for each of these attributes to validate their string value. Figure 5.22 shows the examples of different errors caught in a conditional expression of SACM.

However, Acadela assumes that for HTML, Javascript, and SVG expressions in an Input/OutputField, modelers use a different IDE that validates the syntax and semantically checks the logic of their code through executing it in an external IDE. Thus, modelers only paste a syntactically valid code into the *expression* attribute of the field; hence Acadela does not validate the value's syntax.

5 Language Design

Since Acadela aims to generate compatible CP models for multiple e-Health systems from a single CP definition, including all rules to define the CP concepts of supported e-Health systems will lengthen the base grammar and increase complexity. Therefore, Acadela analyzes the syntax of attributes' value unique to an e-Health system during the interpretation phase. For each unique attribute, the syntax analyzer checks the value using a dedicated sub-grammar, which declares the syntactic rules to parse the value. This results in a shortened and maintainable base Acadela grammar, as the exotic attributes are syntactically analyzed after interpreting CP elements according to the syntax of the client e-Health system.

```
124 OutputField DiastolicAnalysis #left
125     label = 'Diastolic Assessment:'
126     expression = ' if (Diastolic < ) then "Normal"
127                 else if (Diastolic <= 89) then "Elevated"
128                 else "High" '
```

Syntax Error! Invalid expression at line 126:

Expected one of:

1. Number including fraction
2. Integer (Number) at position <absolutePathToGrammarFolder>:(126, 17) => 'astolic < *) then "No'.

```
124 OutputField DiastolicAnalysis #left
125     label = 'Diastolic Assessment:'
126     expression = ' if (Diastolic < 80) : "Normal"
127                 else if (Diastolic <= 89) then "Elevated"
128                 else "High" '
```

Syntax Error! Invalid expression at line 126:

Expected 'then' at position <absolutePathToGrammarFolder>:(126, 20) => 'olic < 80)*: "Normal"'.

Figure 5.22: Example of customized syntax validation for conditional statements in the expression attribute. The top case shows an error of using the wrong keyword (":" instead of "then"); the bottom one demonstrates an incomplete boolean expression. Acadela outputs the corresponding enhanced error messages below the code snippet to state the error cause and solution direction.

5.4.2 Semantic Errors

A syntactically correct CP definition does not guarantee that the CP model is free from logical errors. Therefore, our study develops custom *semantic error analyzers* from scratch to prevent a subset of semantic bugs in the CP definition. Acadela only activates the semantic validator after textX parses the CP definition and generates the corresponding AST. Afterward, Acadela traverses the AST to create a Python class representing the CP metamodel that includes all CP elements with their attributes and line number in the code. This overarching metamodel is the baseline for validating the three categories of semantic errors below. Finally, if an error occurs, Acadela generates an error message stating the *error cause*, *line number*, *erroneous value*, and the *direction to debug*. The Acadela backend returns this enhanced error message to the Acadela IDE to explain the semantic bug to the user.

Non-existing Path or Reference

Several attributes, such as *condition* in Precondition, *expression* in InputField or OutputField, have a String value containing a reference to an (imported) element within the CP. In SACM, this *reference path* starts from the Case hierarchal level for the following attributes: InfoPath in SummaryPanel; *owner* and *dueDateRef* in Stage or Task; *condition* in Precondition. For these attributes, the path pattern is **Setting.<AttributeID>** when referring to a Setting's Attribute, or **<StageID>.<TaskID>.<FieldID>** when referring to an InputField or OutputField element. Acadela semantic error handler verifies if the

5 Language Design

specified path points to an existing Attribute or if the Stage, Task, and Field IDs are correct. The validator throws an error when any part of the reference path is wrong and shows the bug of referring to a non-existing element ID with a line number in the error message, as illustrated in Figure 5.24.

```
1 Stage Evaluation ... // Stage Definition
2
3 HumanTask RequestMedicalTest ... // Task Definition
4 HumanTask MeasureBloodPressure ... // Task Definition
5   Form CgiForm
6     InputField CholesterolTest ... // InputField Definition
7
8 Stage Treatment ... // Stage Definition
9
10 Precondition previousStep = 'Evaluation'
11   // RequestMedicalLabTest does not exist, expect RequestMedicalTest.
12   condition = 'Evaluation.RequestMedicalLabTest.CholesterolTest = 0'
```

Semantic Error: Invalid reference path at line 12! 'RequestMedicalLabTest' Task does not exist. Expect the ID of a defined Task in the Case.

Figure 5.23: Example of invalid referencing path in the condition attribute of a Precondition. At line 12, the *RequestMedicalLabTest* Task ID of the path does not refer to an existing Stage. The intended Task ID is *MeasureBloodPressure*. Finally, Acadela gives a hint that it expects an existing Task ID.

Besides, some attributes should only accept a valid CP element ID as a String value. Declaring an ID that is non-existing or not in the attribute's scope is a semantic error. Attributes of this category are *previousStep* of Precondition that only accepts a valid Stage or Task ID; *expression* of InputField or OutputField that contains references to Input/OutputFields within the same Task. Figure 5.24 and 5.25 show that the violation of these rules generates a semantic error message of an invalid element in *previousStep* and *expression* attributes that reveals the erroneous ID and its line number.

```
1 Stage Evaluation // ... Stage Attributes Definition
2
3 HumanTask RequestMedicalTest ... // Task Definition
4   Form CgiForm
5     InputField CholesterolTest ... // InputField Definition
6
7 HumanTask MeasureBloodPressure ... // Task Definition
8
9 Stage Treatment ... // Stage Attributes Definition
10
11 // CholesterolTest is not the ID of a Stage or Task.
12 Precondition previousStep = 'CholesterolTest'
13   condition = 'Evaluation.RequestMedicalTest.CholesterolTest = 0'
```

Semantic Error at line 12! Stage or Task 'CholesterolTest' does not exist. Expect the ID of an existing Stage or Task.

Figure 5.24: Example of invalid referenced ID in the *previousStep* attribute of a Precondition. At line 11, the *CholesterolTest* ID refers to an InputField but not an existing Stage or Task. The intended ID is *RequestMedicalTest*. Finally, Acadela states that an existing ID of a Stage or Task is required.

```

1 HumanTask MeasureBloodPressure // ... Task Attributes Definition
2
3 Form BloodPressureForm
4   InputField Systolic // ... InputField Definition
5   InputField Diastolic // ... InputField Definition
6
7   OutputField BloodPressureCondition #left
8     label = 'Blood Pressure Assessment:'
9     expression = '
10       if (SystolicValue >= 140 or Diastolic >= 90) then "High"
11       else if (SystolicValue > 120 or Diastolic > 80) then "Elevated"
12       else if (SystolicValue > 90 or Diastolic > 60) then "Normal"
13       else "Low"'

```

Semantic Error at line 10! Invalid field SystolicValue found in the expression of OutputField BloodPressureCondition. Expected the ID of an InputField or OutputField declared in the same Form of OutputField BloodPressureCondition.

Figure 5.25: Example of invalid referenced ID in the expression attribute of an OutputField. At line 10, 11, and 12, the SystolicValue ID does not refer any InputField or OutputField in the same Form. The intended ID is Systolic. Therefore, Acadela suggests to declare an InputField or OutputField within the same Form of the OutputField.

Non-unique Identifier

To distinguish elements in a SACM CP, Acadela enforces the following requirements on identifiers of CP elements:

1. Stage ID is unique across CP elements, i.e., a Stage ID does not match with the ID of any other Stage, Task, InputField, and OutputField.
2. Task ID in a Stage is unique, i.e., no two Tasks in the same Stage have the same ID. However, Tasks in different Stages can share the same ID. Task ID and Input/OutputField ID can also be the same.
3. The ID of InputFields and OutputField are unique within a Form, i.e., no two InputFields or OutputFields in a Form share the same ID. However, InputFields and OutputFields of different Tasks can share the same ID.

The above rules conform with SACM constraints on unique IDs of StageDefinitions, TaskDefinitions, and TaskParamDefinitions. Suppose the CP definition violates one of the above rules. In that case, Acadela shows an error message stating which *element ID* is not unique at which *line* and what *element type(s)* the modelers should check for ID uniqueness. Figure 5.26 and 5.27 show two examples of enhanced error messages of non-unique IDs for Stage and HumanTask, respectively.

```

1 Stage Identification
2 Stage Identification

```

Stage IDs should be unique! Identification at line 1 and column 5 is a duplicate. Please verify that the IDs are unique for each Stage.

Figure 5.26: Example of an enhanced error message for duplicate Stage IDs.

5 Language Design

```
1 Stage MeasureBloodPressure
2   \ \ .. Stage Attributes Definition
3   HumanTask MeasureBloodPressure
```

Task IDs should be unique! *MeasureBloodPressure* at line 3 and column 1 is a duplicate. Please verify that the Task IDs does not match with a Stage ID.

Figure 5.27: Example of an enhanced error message for duplicate Stage and Task IDs.

Untrusted External Service URLs

Medical institutions have strict security policies to filter untrusted inbound or outbound connections. Since SACM supports sending CP data to external medical systems, Acadela should ensure that any SACM CP only sends HttpHook requests to trusted APIs of the partner system. Additionally, Acadela should prevent modelers from sending unauthorized requests to the partner systems' APIs. As a result, Acadela enforces a list of trusted API URLs; each URL has allowed HTTP methods for SACM to use. The URLs specify trusted communication channels with partner systems, while the permitted HTTP methods define the authorized methods for SACM. Figure 5.28 and 5.29 demonstrate two examples of declaring untrusted URL and HTTP method. In these scenarios, the trusted URL is "https://partnersystem.de/monitor" and its authorized HTTP methods are GET and POST.

```
1 Workspace Demo
2
3 define Case Hypertension
4   ... // Case Attributes and child elements Definition
5   Stage Evaluation
6     ... // Stage Attributes Definition
7   Trigger
8     On complete invoke 'http://untrustedparty/api/case' method post
```

The URL `http://untrustedparty/api/case` at line 8 and column 17 is not in the list of trusted sources for workspace Demo. Please check the trusted sources list for the permitted URLs.

Figure 5.28: Example of an enhanced error message for sending requests to untrusted URLs of external services.

```
1 Workspace Demo
2
3 define Case Hypertension
4   ... // Case Attributes and child elements Definition
5   Stage Evaluation
6     ... // Stage Attributes Definition
7   Trigger
8     On complete invoke 'https://partnersystem.de/monitor' method DELETE
```

The URL `https://partnersystem.de/api/monitor` at line 8 does not accept the HTTP method DELETE. Allowed methods for this URL: GET , POST.

Figure 5.29: Example of an enhanced error message for sending requests to trusted URLs but using unauthorized HTTP Method. In this example, the trusted URL does not allow SACM to use the DELETE method.

5.5 Syntax Optimization Effect

By relieving modelers from declaring repetitive or frequently used attributes and shortening system-defined properties (e.g., mandatory, multiplicity), Acadela reduced the effort in defining CP elements. This effect is evident when defining minimal workflow elements such as Stages, Tasks, InputFields, or OutputFields, which require two separate definitions of schema, execution behavior and visual representation definitions in SACM. Figure 5.30 demonstrates the optimized syntax of Acadela (right) from the SACM XML (left) in terms of characters and lines used to define a minimal Stage Definition.

<pre> 1 <EntityDefinition id="Demo_LabTest" 2 description="Lab_Test"> 3 // AttributeDefinitions of Tasks 4 </EntityDefinition> 5 6 <StageDefinition id="Demo_LabTest" 7 isMandatory="false" 8 description="Lab_Test" 9 repeatable="ONCE" 10 entityDefinitionId="Demo_LabTest" 11 entityAttachPath="Demo_LabTest"> 12 // TaskDefinitions 13 </StageDefinition> </pre>	<pre> 1 Stage LabTest 2 #notmandatory 3 label = 'Lab Test' 4 // Define HumanTasks </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------

Figure 5.30: Example definition of a minimum Stage in SACM (left - 279 characters) and Acadela (right - 49 characters)

Considering a complete definition of a CP element, Acadela is similar to SACM in specifying user-defined attributes. However, Acadela's directives and automatic construction of typical attributes' values partially reduce the code size. For example, Figure 5.31 demonstrates a complete HumanTask definition in SACM (left) and Acadela (right). Acadela optimizes the syntax with the following features:

1. Shortens the type definitions of InputFields using directives (text, number(>=0)). For multiple-choice questions, i.e., enumeration type, Acadela removes unnecessary keywords.
2. Automatically constructs the values of *entityDefinitionPath* and *entityAttachPath* attributes.
3. Relieves the definition of *multiplicity*, *mandatory*, and *readOnly* attributes in Input/OutputFields by assigning their default value if not declared by modelers.

Furthermore, since Acadela supports importing CP elements, modelers can define shared CP items used across different CPs. These reusable elements are typically administrative procedures, e.g., patient admission and discharge. The import feature can significantly reduce the CP code size for declaring dynamic templates containing a lengthy SVG image definition; since modelers can include the template in the current CP model definition with two lines of code.

5 Language Design

<pre> 1 <EntityDefinition id = "Demo_Drug" 2 description = "Drug"> 3 <AttributeDefinition id = "drug" 4 type = "string" 5 multiplicity = "exactlyOne" 6 description = "Drug" /> 7 <AttributeDefinition id = "dosage" 8 type = "string" 9 multiplicity = "exactlyOne" 10 description = "Dosage" /> 11 <AttributeDefinition id = "freq" 12 type = "number.min(0)" 13 multiplicity = "exactlyOne" 14 description = "Drug Frequency" 15 defaultValues=" [1]" /> 16 <AttributeDefinition id = "frequit" 17 type = "enumeration" 18 multiplicity = "exactlyOne" 19 description = "Drug Frequency Unit" 20 defaultValues = "['DAYS']"> 21 <EnumerationOption value="DAYS" 22 description = "Days" /> 23 <EnumerationOption value = "WEEKS" 24 description = "Weeks" /> 25 <EnumerationOption value = "MONTHS" 26 description = "Months" /> 27 </AttributeDefinition> 28 29 <HumanTaskDefinition id = "Demo_Drug" 30 isMandatory = "false" 31 repeatable = "PARALLEL" 32 activation = "MANUAL" 33 ownerPath = "Demo_Settings.Clinician" 34 description = "Drug" 35 dynamicDescriptionPath = "Demo_Workplan. 36 Demo_Drug.drug" 37 externalId = "Drug" 38 dueDatePath = "Demo_Settings.WorkplanDueDate" 39 entityDefinitionId = "Demo_Drug" 40 entityAttachPath = "Demo_Workplan.Demo_Drug"> 41 42 <HttpHookDefinition 43 on = "TERMINATE" 44 url = "http://internalsystem/drug/terminate" 45 method = "POST" 46 failureMessage = "Could not terminate 47 prescription on Partner System!" /> 48 49 <TaskParamDefinition isReadOnly = "false" 50 isMandatory = "true" 51 path="Demo_Workplan.Demo_Drug.drug" /> 52 <TaskParamDefinition isReadOnly = "false" 53 isMandatory = "false" 54 path="Demo_Workplan.Demo_Drug.dosage" /> 55 <TaskParamDefinition isReadOnly = "false" 56 isMandatory = "true" 57 path="Demo_Workplan.Demo_Drug.startdate" /> 58 <TaskParamDefinition isReadOnly = "false" 59 isMandatory = "true" 60 path="Demo_Workplan.Demo_Drug.freq" /> 61 <TaskParamDefinition isReadOnly = "false" 62 isMandatory = "true" 63 path="Demo_Workplan.Demo_Drug.frequit" /> 64 </HumanTaskDefinition> </pre>	<pre> 1 HumanTask Drug 2 #notMandatory 3 #repeatParallel 4 #manualActivate 5 owner = "Demo_Settings.Clinician" 6 label = "Drug" 7 dynamicDescriptionRef = " 8 Demo_Workplan.Demo_Drug.drug" 9 externalId = "Drug" 10 dueDateRef = "Settings. 11 WorkplanDueDate" 12 13 Trigger 14 On terminate 15 invoke "http://internalsystem/ 16 drug/terminate" 17 method POST 18 with failureMessage "Could not 19 terminate prescription on 20 Partner System!" 21 22 Form PrescriptionForm 23 #mandatory 24 #notReadOnly 25 26 InputField drug 27 #text 28 label = 'Drug' 29 30 InputField dosage 31 #text 32 #notMandatory 33 label = 'Dosage' 34 35 InputField freq 36 #number(>=0) 37 label = 'Drug Frequency' 38 defaultValues = "[1]" 39 40 InputField frequit 41 #singleChoice 42 Question = 'Drug Frequency 43 Unit' 44 Option "Days" 45 value = "DAYS" 46 Option "Weeks" 47 value = "WEEKS" 48 Option "Months" 49 value = "MONTHS" </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 5.31: Example definition of a HumanTask in SACM (left - 2007 characters), and Acadela (right - 891 characters)

6 Implementation

Based on the CP requirements and language design principles discussed in the previous sections, our study devise the engineering solution for modeling and visualizing CP models as a web application. The architecture shall support a seamless integration of Acadela into the CP modeling features of SACM, while providing a maintainable, flexible, and extensible structure to support CP modeling in various e-Health systems in the future. This section first present the architecture of Acadela, followed by the description of how the DSL compiles a CP model into a SACM-compatible format, detects syntax or semantic errors, and visualizes the generated CP model on the frontend UI.

6.1 Architecture Design

The Acadela-SACM architecture integrates the CP modeling and visualization in Acadela with the adaptive case management execution engine in SACM. The engineering of the CP modeling solution in Acadela requires 1) a **frontend UI** for modelers to *define* and *preview* CP meta-models and 2) **backend component** to *parse*, *interpret*, and *compile* the CP models written in the Acadela DSL to a SACM-compatible JSON format. The Acadela backend uses the SACM API for creating CP meta-model. This API collects the JSON meta-model definition from the request body. Furthermore, the Acadela backend connects to the SACM engine for checking the existence of a User or Group defined in the meta-model.

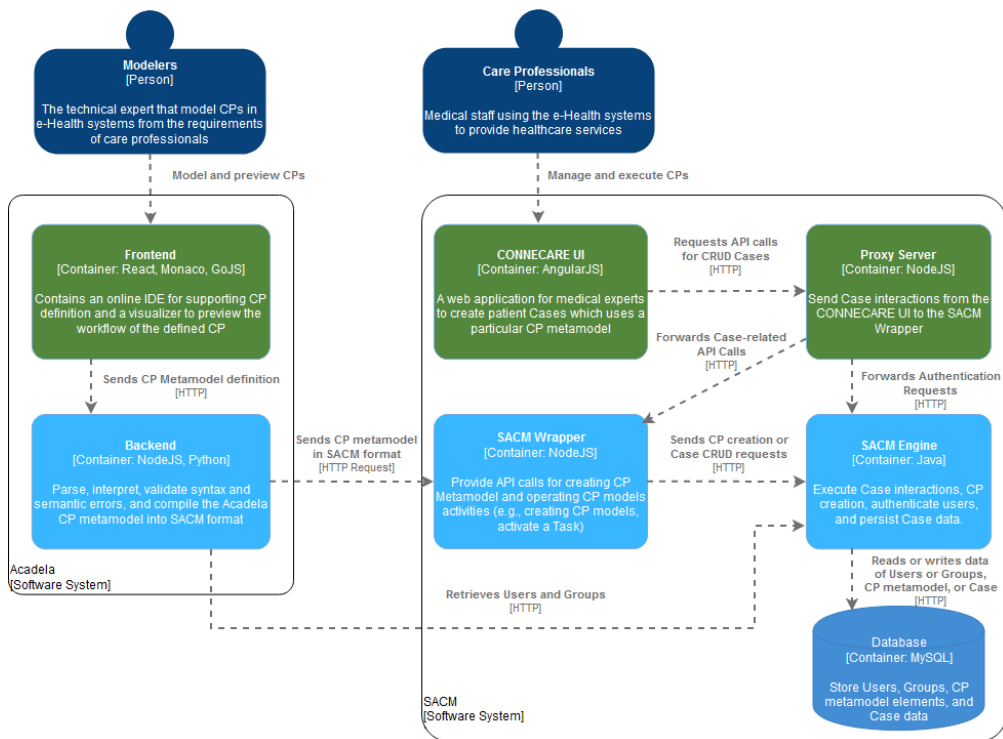


Figure 6.1: Container diagram of the SACM-Acadela system integration. The green containers denote the frontend-related applications, while the blue ones represent the backend applications.

6 Implementation

Once SACM successfully constructs the CP meta-model, medical experts access the CONNECARE UI to create a patient Case that applies the CP. The CONNECARE UI sends all the Case interactions and authentication requests to a proxy server, which forwards the requests to the SACM Wrapper or SACM Engine, respectively. Figure 6.1 illustrates the interactions among subsystems in the Acadela-SACM integrated system through a C4 Container Diagram.

C4 Container Diagram Notations (S. Brown, n.d.): A human shape expresses the user role that interacts with the (sub-)system. A rounded rectangle with white background represents a (sub-)system boundary comprising one or multiple containers. A *Container* in the C4 model is a rounded colored rectangle representing an *executable, independently deployable* application that persists data and performs programmatic instructions. Dotted arrows denote the *relationship* between two containers and their communication purpose.

6.1.1 Acadela System Components

Figure 6.2 and 6.3 demonstrates the architecture of the Acadela **frontend** and **backend** systems using the C4 component diagrams.

C4 Component Diagram Notations (S. Brown, n.d.): The dotted lines represents the currently focused *Container*, which comprise multiple *Components*. A *Component* is a building block of the application, which can represent a *collection of classes*. This class collection can be a module, JAR file, or a library. Each *Component* states the applied technology and its functionality. Dotted arrows denotes the *relationship* between two *Containers* and their communication purpose.

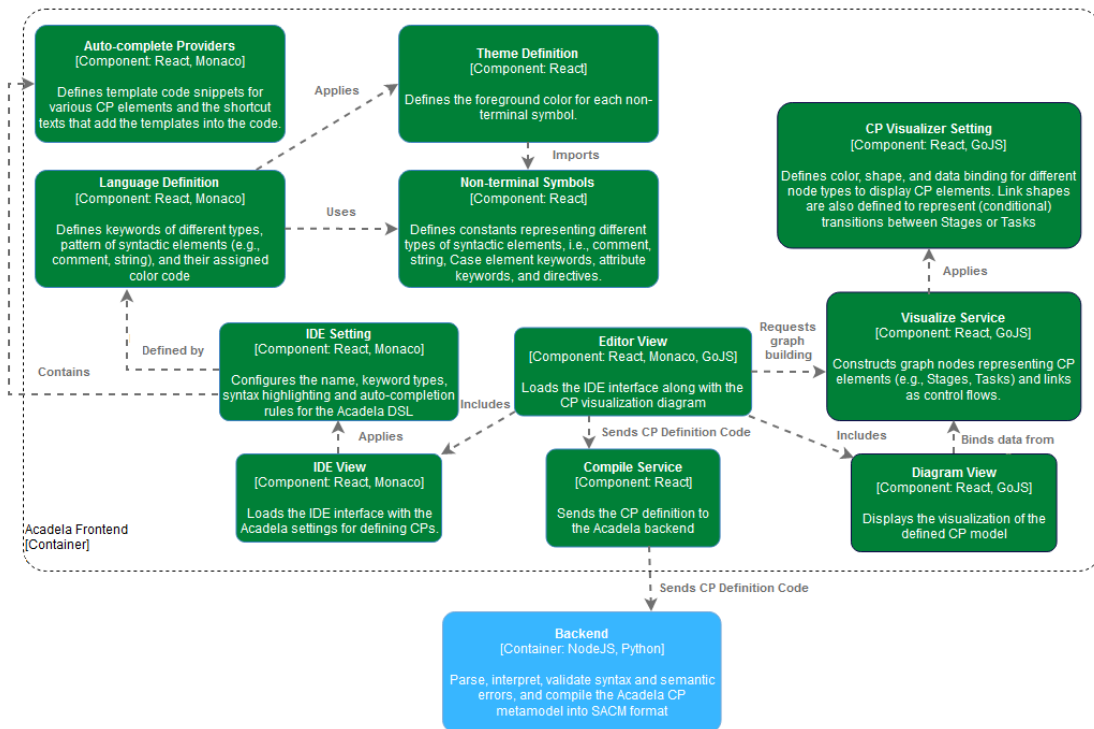


Figure 6.2: Component diagram of the Acadela frontend.

Acadela Frontend The Acadela frontend container constructs the online IDE and a visualizer for modeling and previewing CP meta-models. Acadela leverages the Monaco (Microsoft, 2022) library to define the *IDE layout*, *keywords*, and *rules* for syntax highlighting and auto-completion. An *Editor View*

6 Implementation

React component holds the *IDE View* and CP visualization *Diagram View*. This *Editor View* calls the compile service to send the CP model code written in this IDE to the backend when the modeler clicks a **Submit** or **Validate** button. The response to this request is a message stating whether the code is compiled successfully or an error occurs, which is shown in a *status panel*.

Additionally, the backend also returns a meta-model definition in JSON format. The *Diagram View* uses this JSON meta-model definition and the GoJS library (Northwoods Software, 2022) to construct 1) graph *node* elements with color coding to represent CP elements and 2) *link* elements between *nodes* to represent the control flow of the CP.

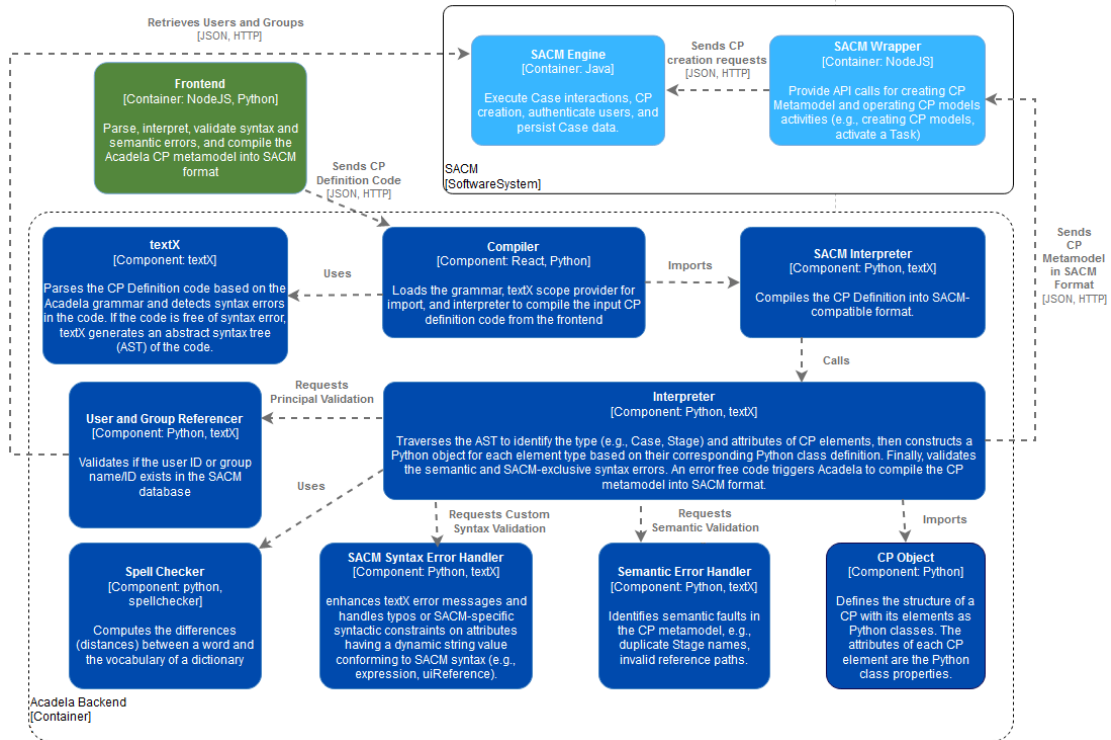


Figure 6.3: Component diagram of the Acadela backend. The dark blue components represent classes and libraries used in the Acadela backend. The light blue rounded rectangles are SACM Containers that the Acadela backend communicates with, and the green container is the Acadela frontend React application.

Acadela Backend The Acadela backend is responsible for *parsing*, *interpreting*, *validating* syntax and semantic errors, and *compiling* the defined CP meta-models into SACM JSON format. Furthermore, the interpreter includes 1) the *textX parser* that performs syntax checking and generates an AST from the CP meta-model code, 2) A *textX meta-model* storing the hierarchical structure and attribute of CP elements as Python classes, 3) A Python SACM *interpreter* that traverses the AST to identify and construct CP elements defined in the CP meta-model by extracting the type and attributes of nodes in the AST. Since the CP hierarchy and attributes are accessible in the *interpreter*, it also validates the SACM-exclusive *string patterns* and semantic constraints. Finally, the Acadela backend compiles the interpreted Python classes of CP meta-model elements into a SACM-compatible CP meta-model in JSON.

One design concern for the Acadela backend is that various e-Health systems may have different syntaxes for declaring an attribute value. For example, *conditional expressions* in SACM are similar to *if-else statements* of the Bash shell, but other e-Health systems may apply a Python or Java syntax. Therefore, Acadela provides two constructs to relieve modelers of other e-Health systems from learning Acadela *conditional expressions*. First, a compile mode *directive* at the beginning of the CP definition file indicates the target e-Health systems. This directive instructs Acadela to select the interpreter for generating the CP model compatible with the target e-Health system. Second, each *interpreter* of an e-Health system

6 Implementation

has a dedicated grammar for validating the syntax of attributes whose values can be expressed differently in various e-Health systems. This design helps Acadela chooses the correct syntax validation mechanism for a particular e-Health application without complicating the CP grammar.

The following subsections describe the procedure of compiling a CP definition from Acadela code to SACM meta-model and visualizing the generated CP. The processes start with modelers defining CP meta-models in the IDE, then sending it to the Acadela backend, which parses, interprets, checks semantic and syntax errors, and compiles the CP model into SACM-compatible JSON format if the code is error-free. Next, Acadela sends the final code to the SACM wrapper, which forwards the CP to the SACM engine for validation and persistence. Simultaneously, the Acadela backend returns the CP in SACM JSON format to the Acadela frontend for constructing the CP workflow visualization.

6.2 Integrated Development Environment (IDE)

Acadela leverages the Monaco IDE to support modelers in defining and debugging CPs with *syntax highlighting* and *auto-completion*. The content of the CP meta-model defined in this IDE is sent to the backend upon clicking the **Validate** or **Submit** button. Figure 6.5 shows the UI of the Acadela IDE.

Auto-completion Acadela leverages two auto-complete types of Monaco: **Text** and **Snippet**, to automatically generate a *keyword* or CP element definition *template*. Each auto-completion requires a rule definition, specifying 1) a *label* that hints at the auto-completed element. Monaco renders the label as an item of a drop-down list when modelers type characters contained in the *label*. In Acadela, this *label* can be the *keyword* or a *description* of a CP element; 2) the auto-complete *type* (**Text** or **Snippet**); 3) the text to be inserted when the user clicks on the *label* option. The auto-completion of **Snippet** requires an additional *insertTextRules* attribute, which Acadela sets as **InsertAsSnippet** for the auto-completion of all CP elements. Figure 6.4 demonstrate the auto-completion of a CP elements in the IDE (left and middle) and their corresponding rule definition (right).

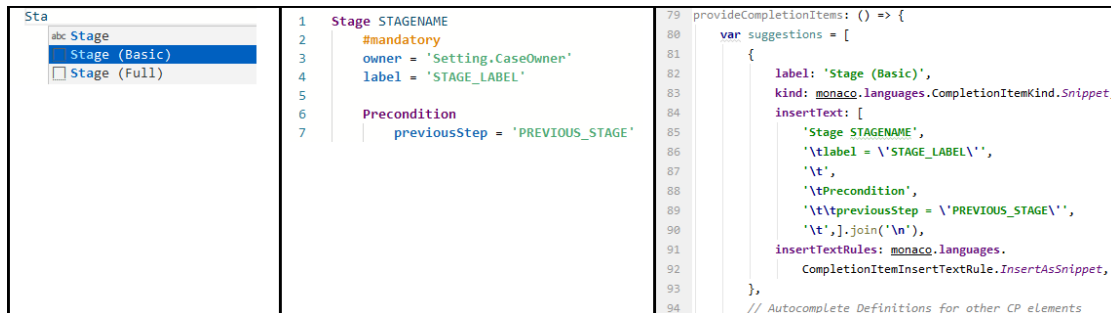


Figure 6.4: Demonstration of Autocomplete in Acadela IDE. The left picture shows an incomplete *keyword* and *suggestions*. The middle one shows the inserted code snippet after selecting the auto-complete option. Finally, the right picture shows the definition of a **Snippet** auto-complete definition in Monaco.

Syntax Highlighting Monaco provides a language definition infrastructure to define a language's *keywords* as **strings** and *code sections* (e.g., comments, strings) as **regular expressions**. Acadela contains three keyword groups: CP element *type*, CP element *attributes*, and *directives*, combined with two *code section* patterns: double/single-quoted *strings* and *comments*. A **tag** (aka. **token** in Monaco) labels each keyword *group* or *code section* pattern. To enable syntax highlighting, Monaco supports a *theme* definition that specifies rules for applying color and text decoration (e.g., bold) to each **tag**. Figure 6.6 illustrates how to define colors (top) to paint *keywords* and *code patterns* (bottom) in Monaco.

```

129 Stage MedicalTest
130   label = 'Medical Test'
131   owner = 'Setting.Nurse'
132
133   Precondition
134     previousStep = 'Evaluation'
135     condition = 'Evaluation.RequestMedicalTest.CholesterolTest = 1'
136
137   Trigger
138     On complete invoke 'http://partnersystem/api' method POST
139     with failureMessage 'Cannot send data to the partner system'
140
141   HumanTask MeasureBloodCholesterol
142     owner = 'Setting.Nurse'
143     label = 'Record Blood Cholesterol'
144
145     Form CholesterolTestForm
146       InputField CholesterolLvl #number #left
147         label = "Total Blood Cholesterol Level (mm/L):"
148         uiRef = 'colors(0<green<=5<red<30)'
149
150       OutputField TotalCholesterolAnalysis #left
151         label = 'Total Blood Cholesterol Assessment:'
152         expression = ' if (DhCholesterolLvl <= 5) then "Normal"
153                    | else "High"'
154
155   Stage Treatment
156     owner = 'Setting.Clinician'
157     label = 'Treatment'

```

Validate Submit

Semantic Error at line 152! Invalid field DhCholesterolLvl found in the expression of OutputField TotalCholesterolAnalysis. Expected the ID of an InputField or OutputField declared in the same Form as OutputField TotalCholesterolAnalysis.

Figure 6.5: The IDE GUI of Acadela with *syntax highlighting*. The top panel shows the code written by modelers. The **Validate** button sends the code to the backend for verifying syntactic or semantic errors. The **Submit** button has the same feature as the **Validate** one, but the backend further checks the existence of defined Users and Groups in the CP and sends the CP meta-model to SACM. Finally, the bottom panel shows the status of the code compilation or an enhanced error message if a bug occurs.

Code Submission When clicking the **Validate** or **Submit** button, the IDE sends the code to the backend. A *validation* request only asks the backend to check for syntax and semantic errors in the code and returns a CP meta-model in JSON format to the frontend. A *submission* request executes the actions of *validation* requests, plus sending the CP meta-model to SACM for creating the executable CP.

For the *validation* request, the Acadela backend returns an enhanced error message if it detects a syntax or semantic error in the code. A *submission* request may receive an **Internal SACM error** when a bug occurs within the SACM system or a non-existing Group or User defined in the Responsibilities element.

The following section describes how Acadela applies the grammar to parse the CP meta-model definition.

6 Implementation

```
3 static themeDef = {
4   base: 'vs',
5   inherit: false,
6   rules: [
7     { token: NonTerminalSymbol.OBJECT, foreground: '660066' /*purple*/, fontStyle: 'bold' },
8     { token: NonTerminalSymbol.ATTRIBUTE, foreground: '1167b1' /*dark blue*/, fontStyle: 'bold' },
9     { token: NonTerminalSymbol.DIRECTIVE, foreground: 'E08400' /*orange*/, fontStyle: 'bold' },
10    { token: NonTerminalSymbol.ERROR, foreground: 'b4151c' /*red*/, fontStyle: 'bold' },
11    { token: NonTerminalSymbol.STRING, foreground: '4ca973' /*green*/, fontStyle: 'bold' },
12    { token: NonTerminalSymbol.COMMENT, foreground: 'bebebe' /*grey*/, fontStyle: 'bold' },
13    { token: NonTerminalSymbol.DEFAULT, foreground: '014666' /*light blue*/ }
14  ]
}

5 static LanguageDef = {
6   cpElementKeywords: [
7     'Workspace', 'Case', /* Other CP element keywords */, 'OutputField'
8   ],
9   attrKeywords: [
10    'multiplicity', 'expression', /* Other attributes keywords */, 'activate'
11  ],
12  directiveKeywords: [
13    'any', 'notMandatory', /* Other directive keywords */, 'noRepeat', 'repeatSerial'
14  ],
15  tokenizer: {
16    root: [
17      [/[a-z_$][\w$]*/, {
18        cases: {
19          '@cpElementKeywords': NonTerminalSymbol.CPELEMENT,
20          '@attrKeywords': NonTerminalSymbol.ATTRIBUTE,
21          '@default': NonTerminalSymbol.DEFAULT
22        }
23      }],
24
25      [/\#[\w\.\.]*[\\(\w-\\)]*/, { token: NonTerminalSymbol.DIRECTIVE } ],
26
27      [/"*/, { token: NonTerminalSymbol.STRING, bracket: '@open', next: '@stringDoubleQuote' } ],
28      [/'*/, { token: NonTerminalSymbol.STRING, bracket: '@open', next: '@stringSingleQuote' } ],
29
30      { include: '@whitespace' }
31    ],
32    comment: [
33      [/^\/\*/, NonTerminalSymbol.COMMENT],
34      [/\//, NonTerminalSymbol.COMMENT, '@push'], // nested comment
35      ["/\*/, NonTerminalSymbol.COMMENT, '@pop'],
36      [/\//, NonTerminalSymbol.COMMENT]
37    ],
38    stringDoubleQuote: [
39      [/"*/, NonTerminalSymbol.STRING],
40      [/\./, 'string.escape.invalid'],
41      [/'*/, { token: NonTerminalSymbol.STRING, bracket: '@close', next: '@pop' } ]
42    ],
43  }
}
```

Figure 6.6: Syntax highlighting definition for *keywords* and *code sections* in Monaco (bottom) based on a set of color rules (top). Lines 27 and 28 instructs Monaco to identify multi-line strings in the code.

6.3 Grammar Definition

The Acadela grammar resides on the backend to define the abstract syntax, which contains rules declaring constructs of CP elements. Each rule specifies the *attribute*, *child element*, or *string literal* used in the CP element. Applying the grammar rules to a CP definition code helps textX identify syntax errors and

generate an abstract syntax tree (AST) to represent the structure of the code elements. The following subsections consecutively explain 1) the *rule expressions* used in the grammar, 2) examples of *construct definitions* using the *rules*, and 3) the *generated AST* of a CP meta-model definition code.

6.3.1 textX Grammar Rule Expressions

Each rule starts with a string *name* and a *colon*, then the *body*, and ends with a *semicolon*. textX grammar (Dejanović, n.d.b) introduces the following *types* and *operators* to define rules.

String Match: a double- or single-quoted String. Acadela uses this rule to define its operators which accept whitespaces (e.g., space or new line) before or after them. For example, Acadela expresses an equal sign ("=") as follows:

```
1 Eq :
2   ' = '
3   ;
```

This rule has the name "Eq" and allows modelers to state either "[attributeName]>=<value>" or "<attribute> = <value>" in their code.

Regex Match: applies a Python regular expression wrapped inside slashes, i.e., `/<regex_rule>/`. textX uses the python `re` module internally to verify regular expressions matching. Acadela uses **Regex Match** to declare keywords that a whitespace must follow. For example, the rule definition for the keyword "Stage" is as follows:

```
1 StageTerm :
2   /(Stage)\s/
3   ;
```

The parentheses represent a regex group, which specifies the text *Stage* in this case. Additionally, the `\s` pattern represents a whitespace. As a result, stating "Stage <StageId>" is valid, but "Stage<StageId>" throws a syntax error as no whitespace exists after the *Stage* keyword.

Repetitions: defines constraints on the *multiplicity* of a matched expression. textX grammar provides the following three repetition rules and their operator.

1. **Zero or more (*):** An expression can be **undefined** or appear **multiple times**. Acadela uses this operator to define *optional CP concepts* having more than one sibling, e.g., a Precondition can have multiple *previousStep* definitions to denote the conjunction (AND) relationship among them, as shown in line 4 of the following rule:

```
1 Precondition :
2   PreconditionTerm
3   (
4     ('previousStep' Eq stepList += STRING)*
5     ('condition' Eq entryCondition = STRING)?
6   )#
7   ;
```

2. **One or more (+):** An expression must appear **at least once**. Acadela leverages this rule to define a pattern that must have at least one repetition. For example, a color code definition in the *uiRef* attribute, e.g., "0<green<=80<yellow<=89<red<300", has a number followed by at least one or more color bands defined as a "Comparator ColorName Comparator NUMBER" sequence. **Comparator** refers to the following operators: `=, <>, <=, >=, <, >`. The below Stage rule snippet shows how to enforce this constraint in line 8.

6 Implementation

```
1 Comparator:
2   '=' | '<>' | '<=' | '>=' | '<' | '>'
3 ;
4 ColorName:
5   'red' | 'blue' | 'green' | 'orange' | 'yellow'
6 ;
7 CompareExpression:
8   NUMBER (Comparator ColorName Comparator NUMBER)+
9 ;
```

3. **Unordered group (#):** Expressions can appear in any **arbitrary sequence**. Thanks to this rule, *child elements*, *directives*, or *attributes* of any CP object in Acadela can appear **in any order**. For example, the following Stage grammar snippet illustrates how the group of Stage *directives* (WorkflowElementDirective) can appear in any order using the # operator in line 7.

```
1 WorkflowElementDirective:
2   (
3     (mandatory = Mandatory)?
4     (repeatable = Repeatable)?
5     (activation = Activation)?
6     (multiplicity = Multiplicity)?
7   )#
8 ;
9
10 Stage:
11 (
12   StageTerm name = ID
13   directive = WorkflowElementDirective
14   (
15     // Other Stage attributes and child elements rules
16   )#
17 );
```

Assignments: textX supports creating object's *attributes* in grammar rules. During the meta-model construction process, textX creates a Python class for each rule along with their *attributes* defined in the grammar.

Each *attribute* assignment has a left-hand side (LHS) and a right-hand side (RHS). The LHS declares the *attribute*'s name, while the RHS determines the *type*. If the RHS is a **BASETYPE** of textX, i.e., **STRING**, **ID**, **BOOLEAN**, **NUMBER**, **INT**, **FLOAT**, then enforcing a Python primitive type (e.g., int, string, boolean). Otherwise, the RHS refers to the *matched rule*. If the RHS is a regular expression, textX assigns the string matched by the expression as the RHS value. Acadela uses the following assignment operators of textX.

1. **Plain assignment (=):** assigned the matched rule defined in the RHS to the LHS attribute **once**. Acadela uses **plain assignment** to declare *keywords*, *directives*, *CP elements*, or *attributes* that only accept **one matched rule expression**, e.g., **#mandatory**, *label*, *dueDateRef*, *Form*.
2. **Zero or more assignment (*=):** An attribute is a **list that can be empty**. Acadela uses this operator to define *optional CP elements* with **zero or at least one sibling**, e.g., *Preconditions* in a *Stage*. A *Stage* can have zero or at least one *Precondition*, as enforced by the following rule:

```
1 Stage:
2 (
3   StageTerm name = ID
4   directive = WorkflowElementDirective
5   (
6     // Other stage attributes
7     (preconditionList *= Precondition)
```

6 Implementation

```
8     // Hook and Task(s) Rules
9     )#
10 );
```

3. **One or more assignment (+=):** An attribute is a **list** with **at least one item**. Acadela leverages this rule to define mandatory CP elements that allow **multiple siblings**. For example, a Stage has multiple Tasks, but every valid Stage must contain at least one Task. The below Stage rule snippet shows how to enforce this constraint in line 7.

```
1 Stage:
2 (
3     StageTerm name = ID
4     directive = WorkflowElementDirective
5     (
6         // Other stage attributes
7         taskList += Task
8         // Precondition and HttpHook Rules
9     )#
10 );
```

Optional: Instructs textX to match an expression if possible but does not throw an error for unmatched cases. Placing a `?` operator at the end of a statement triggers the optional rule. Acadela uses this rule to define **non-mandatory attributes** or *CP concepts* in a CP element. The below example shows that a Stage can contain an optional *additionalDescription* attribute and a Trigger section containing HttpHook definitions.

```
1 Stage:
2 (
3     StageTerm name = ID
4     directive = WorkflowElementDirective
5     (
6         // Other stage attributes
7         (additionalDescription = AdditionalDescription)?
8         (TriggerTerm hookList *= HttpHook)?
9         // Precondition and Task(s) Rules
10    )#
11 );
```

Ordered choice: To specify **alternative rules** at a particular position, textX provides a `|` operator to express the *disjunction (OR) relationship* among the rules. Acadela leverages the **ordered choice** to define *possible options* of a rule. For example, the following grammar snippet states that the BasicArithmetic-Operator construct can contain one of the *plus*, *minus*, *multiplication*, and *division* symbols.

```
1 BasicArithmeticOperators:
2     '+' | '-' | '*' | '/'
3 ;
```

textX maps the expressions in the **ordered choice** from left to right; thus, it takes the first matching expression.

Another usage of **ordered choice** is for **importing elements**. Acadela leverages the textX import by reference to state that a CP element can be defined within the same grammar or imported from an external file. The following example shows that a Stage rule accepts an internal definition or an imported Stage element (FQN) satisfying the Stage's constraints. The FQN states that textX finds the imported Stage by its ID (See **Import Scoping Definition** in Section 6.4).

6 Implementation

```
1 Ref:
2   /(use)\s/
3 ;
4 RefStage:
5   Ref StageTerm
6 ;
7 Stage:
8   (
9     StageTerm name = ID
10    directive = WorkflowElementDirective
11    (
12      // Stage attributes and child elements
13    )#
14  )
15  // Import element with command "use Stage <StageID>"
16  // or "<importAlias>.<StageID>"
17  | (RefStage ref=[Stage|FQN])?
18 ;
```

In line 17, `ref` is a textX attribute to declare a *reference* for a given construct. The first part states the rule name to verify the imported object syntactically. The second part is a (FQN) (FullyQualifiedName) that defines the location of the referred object, which is typically the `name` attribute's value in a grammar rule. Since Acadela uses this `name` attribute as the CP element ID, all imported CP items are referable by their ID.

6.3.2 Grammar Specification for Modeling Clinical Pathways

Acadela leverages the textX rules described in the previous subsection to define the **terminals** and **non-terminals** in its CP modeling grammar. **Terminals** are symbols that construct the alphabet of a language (Shinan, 2020). They are **leaf-level rules**; hence the grammar derivation terminates at their position (Umrigar, 1997). **Terminals** in Acadela are string, number, alphabet characters, and arithmetic symbols (e.g., "+", "/", "<").

A **nonterminal** is a language part consisting of **terminals** or **productions**, which are rules defining how to expand a **nonterminal** in the LHS with **terminals** or **nonterminals** on the RHS (Fender, 2018). Acadela uses **nonterminals** to declare *CP modeling concepts*, their *attributes*, and *child elements*. Listing 6.1 shows an excerpt of the Acadela grammar that contains representative **nonterminals**.

The entire grammar in Appendix AZ contains other **nonterminal** rules specifying similar **terminal** patterns. For example, `StageTerm` and `TaskTerm` **nonterminals** define a regular expression of the keyword ("*Stage*" or "*Task*"), thus the excerpt only includes `StageTerm`. The next subsection reflects on the grammar's capability to address CP modeling requirements presented in Section 5.1.

```
1 Start:
2   (versionTag = AcaVersion)?
3   (importList *= Import)?
4   (
5     (defWorkspace = DefWorkspace)
6     | ((define)\s/ objList *= Obj)*
7   )
8 ;
9
10 Obj:
11   ( Case | CaseSetting | Stage | Task | Form | InputField |
12     OutputField | Hook | AttributeValue)
13 ;
14
15 AcaVersion: /(#aca)\d\.\d/ ;
```


6 Implementation

```
16
17 // Define import by ID and support Import from file.
18 FQN: ID+['.'];
19 FQNI: ID+['.']*(['.*'])?;
20
21 // Import Definition Level
22 Import:
23   'import' importURI=FQNI ('as' name=ID)?
24 ;
25
26 DefWorkspace:
27   workspace = Workspace
28   /(define)\s/ case = Case
29 ;
30
31 Workspace:
32   WorkspaceTerm BasicIdentity
33 ;
34
35 BasicIdentity:
36   name = ID
37   ('staticId' Eq staticId=STRING)?
38 ;
39
40 GroupIdentity:
41   name = ID
42   (
43     ('staticId' Eq staticId=STRING)?
44     ('name' Eq groupName=STRING)
45   )#
46 ;
47
48 /*****
49 ***** CASE *****/
50 *****/
51
52 Case:
53 (
54   CaseTerm name=ID
55   (
56     casePrefix = CasePrefix
57     (
58       ('version' Eq version = INT)
59       description = Description
60       responsibilities = Responsibilities
61       setting = CaseSetting
62       summary = SummaryPanel
63       ('Trigger' hookList += CaseHook)?
64       (entityDefinitionId = STRING)?
65       (entityAttachPath = STRING)?
66       (notes = STRING)?
67       stageList += Stage
68     )#
69   )#
70 ) | (Ref /(Case)\s/ ref=[Case|FQN])?
71 ;
72
73 Responsibilities:
74   /(Responsibilities)\s/
75   (
76     groupList *= Group
```

6 Implementation

```
77     userList *= User
78   )#
79 ;
80
81 Group: GroupTerm GroupIdentity ;
82
83 User: UserTerm BasicIdentity ;
84
85 CaseSetting:
86 (
87   SettingTerm
88   (description = Description)?
89   (
90     caseOwner = CaseOwner
91     (attrList *= Attribute)
92     (casePatient = CasePatient)?
93     (attrList *= Attribute)
94   )#
95 ) | (RefSetting ref=[CaseSetting|FQN])?
96 ;
97
98 CaseOwner:
99   /(CaseOwner)\s/ group = TextNoQuote
100   attrProp = AttributeProp
101 ;
102
103 CasePatient:
104   /(CasePatient)\s/ group = TextNoQuote
105   attrProp = AttributeProp
106 ;
107
108 /*****
109 ***** STAGE *****
110 *****/
111
112 Stage:
113 (
114   StageTerm name = ID
115   directive = WorkflowElementDirective
116   (
117     (description = Description)
118     (ownerPath = OwnerPath)?
119     (clientPath = ClientPath)?
120     (dynamicDescriptionPath = DynamicDescriptionPath)?
121     (externalId = ExternalId)?
122     (additionalDescription = AdditionalDescription)?
123   )#
124   (
125     (preconditionList *= Precondition)
126     (TriggerTerm hookList *= HttpHook)?
127     taskList += Task
128   )#
129 ) | (RefStage ref=[Stage|FQN])?
130 ;
131
132 /*****
133 ***** TASK *****
134 *****/
135
136 Task: HumanTask | AutomatedTask | DualTask ;
137
```

6 Implementation

```
138 HumanTask:
139 (
140   HumanTaskTerm name = ID
141   directive = WorkflowElementDirective
142   attrList = SharedTaskAttrs
143   (
144     (TriggerTerm hookList *= HttpHook)?
145     form = Form
146   )#
147 ) | (RefTask ref=[HumanTask|FQN])?
148 ;
149
150 AutomatedTask:
151 (
152   AutoTaskTerm name = ID
153   directive = WorkflowElementDirective
154   attrList = AutomatedTaskAttrs
155   (
156     (TriggerTerm hookList *= HttpHook)?
157     form = Form
158   )#
159 ) | (RefTask ref=[AutomatedTask|FQN])?
160 ;
161
162 DualTask:
163 (
164   DualTaskTerm name = ID
165   directive = WorkflowElementDirective
166   attrList = SharedTaskAttrs
167   (
168     (TriggerTerm hookList *= DualTaskHttpHook)?
169     form = Form
170   )#
171 ) | (RefTask ref=[DualTask|FQN])?
172 ;
173
174 AutomatedTaskAttrs:
175 (
176   description = Description
177   (ownerPath = OwnerPath)?
178   (externalId = ExternalId)?
179   (dynamicDescriptionPath = DynamicDescriptionPath)?
180   (additionalDescription = AdditionalDescription)?
181   (preconditionList *= Precondition)
182 )#
183 ;
184
185 SharedTaskAttrs:
186 (
187   description = Description
188   (ownerPath = OwnerPath)?
189   (dueDatePath = DueDatePath)?
190   (externalId = ExternalId)?
191   (additionalDescription = AdditionalDescription)?
192   (dynamicDescriptionPath = DynamicDescriptionPath)?
193   (preconditionList *= Precondition)
194 )#
195 ;
196
197
198
```

6 Implementation

```
199 Precondition:
200   PreconditionTerm
201   (
202     ('previousStep' Eq stepList += STRING)*
203     ('condition' Eq entryCondition = STRING)?
204   )#
205 ;
206
207 /*****
208 ***** FORM *****
209 *****/
210
211 Form:
212 (
213   FormTerm name = ID
214   (directive = FormDirective)?
215   fieldList += FormField
216 ) | (RefForm ref=[Form|FQN])?
217 ;
218
219 FormDirective:
220 (
221   (mandatory = Mandatory)?
222   (readOnly = ReadOnly)?
223 )#
224 ;
225
226 FormField:
227   InputField/[\s\n]*/
228   | OutputField/[\s\n]*/
229 ;
230
231 InputField:
232 (
233   InputFieldTerm name = ID
234   directive = InputFieldDirective
235   (
236     (
237       description = Description
238       | question = Question
239     )
240     (path = CustomElementRefPath)?
241     (uiRef = UiReference)?
242     (externalId = ExternalId)?
243     (additionalDescription = AdditionalDescription)?
244     (defaultValue = DefaultValue)?
245     (defaultValues = DefaultValues)?
246   )#
247 ) | (Ref InputFieldTerm ref=[InputField|FQN])?
248 ;
249
250 CustomElementRefPath: 'ElementPath' Eq value=STRING ;
251
252 Question:
253   'Question' Eq text=STRING
254   optionList += Option
255 ;
256
257 Option:
258   /(Option)\s/ (
259     (key=STRING)
```

6 Implementation

```
260     ('value' Eq value=STRING)
261     (additionalDescription = AdditionalDescription)?
262     (externalId = ExternalId)?
263   )#
264 ;
265
266 OutputField:
267 (
268   OutputFieldTerm name = ID
269   directive = OutputFieldDirective
270   (
271     description = Description
272     (additionalDescription = AdditionalDescription)?
273     (uiRef = UiReference)?
274     (path = CustomElementRefPath)?
275     (expression = OutputFieldExpression)?
276     (externalId = ExternalId)?
277     (defaultValue = DefaultValue)?
278     (defaultValues = DefaultValues)?
279   )#
280 ) | (Ref OutputFieldTerm ref=[OutputField|FQN])?
281 ;
282
283
284 /*****
285 ***** SETTING ATTRIBUTE SECTION *****
286 *****/
287
288 Attribute:
289   AttributeTerm name = ID
290   attrProp = AttributeProp
291 ;
292
293 AttributeProp:
294   directive = AttributeDirective
295   (
296     description = Description
297     (externalId = ExternalId)?
298     (additionalDescription = AdditionalDescription)?
299     (uiRef = UiReference)?
300     (defaultValue = DefaultValue)?
301     (defaultValues = DefaultValues)?
302   )#
303 ;
304
305 AttributeDirective:
306   (multiplicity = Multiplicity)?
307   (type = Type)?
308 ;
309
310 /*****
311 ***** SUMMARY SECTION *****
312 *****/
313
314 SummaryPanel: /(SummaryPanel)\s/ sectionList += SummarySection ;
315
316 SummarySection:
317   /(Section)\s/ name = ID
318   (directive = SummarySectionPosition)?
319   description = Description
320   paramList += SummaryParam
```

6 Implementation

```
321 ;
322
323 SummaryParam:
324   /(InfoPath)\s/ path = TextNoQuote
325 ;
326
327 /*****
328 ***** HTTPHOOK *****
329 *****/
330
331 Hook: CaseHook | HttpHook | DualTaskHttpHook ;
332
333 CaseHook:
334 (
335   /(Hook)\s/ name = ID)?
336   /(On)\s/ event = CaseHookEvent
337   /(invoke)\s/ url = STRING
338 ) | (RefHook ref=[CaseHook|FQN])?
339 ;
340
341 HttpHook:
342 (
343   /(Hook)\s/ name = ID)?
344   /(On)\s/ event = BaseEvent
345   (
346     /(invoke)\s/ url = STRING)
347     /(method)\s/ method = HttpMethod)
348   /(with failureMessage)\s/ failureMessage = STRING)?
349 )#
350 ) | (RefHook ref=[HttpHook|FQN])?
351 ;
352
353 DualTaskHttpHook:
354 (
355   /(Hook)\s/ name = ID)?
356   /(On)\s/ event = DualTaskEvent
357   (
358     /(invoke)\s/ url = STRING)
359     /(method)\s/ method = HttpMethod)
360   /(with failureMessage)\s/ failureMessage = STRING)?
361 )#
362 ) | (RefHook ref=[DualTaskHttpHook|FQN])?
363 ;
364
365 SharedEvent:
366   'available' | 'enable' | 'activate' | 'complete' | 'terminate'
367 ;
368
369 CaseHookEvent: SharedEvent | 'delete' ;
370
371 BaseEvent: SharedEvent | 'correct' ;
372
373 DualTaskEvent:
374   'activatehumanpart' | 'activateautopart' | 'completehumanpart'
375   | 'completeautopart' | 'correcthumanpart' | 'correctautopart'
376   | BaseEvent
377 ;
378
379 CasePrefix: 'prefix' Eq value = STRING ;
380
381 Description: ('label' Eq value = STRING) ;
```

6 Implementation

```
382
383 /*****
384 ***** DIRECTIVES *****
385 *****/
386
387 InputFieldDirective:
388 (
389   (mandatory = Mandatory)?
390   (readOnly = ReadOnly)?
391   (position = Position)?
392   (multiplicity = Multiplicity)?
393   (part = Part)?
394   (type = FieldType)?
395 )#
396 ;
397
398 OutputFieldDirective:
399 (
400   (mandatory = Mandatory)?
401   (readOnly = ReadOnly)?
402   (position = Position)?
403   (explicitType = Type)?
404 )#
405 ;
406
407
408 WorkflowElementDirective:
409 (
410   (mandatory = Mandatory)?
411   (repeatable = Repeatable)?
412   (activation = Activation)?
413   (multiplicity = Multiplicity)?
414 )#;
415
416 Multiplicity:
417   Hash ( 'maxOne' | 'exactlyOne' | 'atLeastOne' | 'any' )
418 ;
419
420 Type:
421   Hash (
422     LinkType | DocumentLinkType | 'notype' | 'text'
423     | 'longtext' | 'string' | 'boolean' | NumType
424     | 'singlechoice' | 'multiplechoice'
425     | DateType | 'json' | 'custom'
426   );
427
428 FieldType: Type ;
429
430 LinkType:
431   'link' '.' (linkType='Users' | linkType='Entity')
432   '( ' linkObj += TextNoQuote (',' linkObj += TextNoQuote)? ' )'
433 ;
434
435 DocumentLinkType: 'documentlink' '( ' url=STRING ' )' ;
436
437 DateType:
438   'date.after(TODAY)'
439   | /(date)\s/
440 ;
441
442
```

6 Implementation

```
443 // support number, number(<Comparator> INT), number(INT-INT)
444 NumType:
445   'number' (
446     '('
447     ( (comparator=Comparator num=INT) | (min=INT '-' max=INT) )
448     ')',
449   )?
450 ;
451
452 Part:
453   Hash ( 'humanDuty' | 'systemDuty' )
454 ;
455
456 Repeatable:
457   Hash ( 'repeatSerial' | 'noRepeat'
458         | 'repeatParallel' ( '(' INT ')' )? )
459 ;
460
461 Mandatory: Hash ( 'mandatory' | 'notmandatory' ) ;
462
463 Activation:
464   Hash ( 'manualActivate' | 'autoActivate'
465         | 'activateWhen' '(' STRING ')' )
466 ;
467
468 ReadOnly: Hash ( 'readOnly' | 'notReadOnly' ) ;
469
470 SharedPosition: 'stretched' | 'left' | 'center' | 'right';
471
472 SummarySectionPosition: Hash SharedPosition ;
473
474 Position:
475   Hash ( 'leftcenter' | 'centerright' | SharedPosition )
476 ;
477
478 /*****
479 ***** SHARED ATTRIBUTES *****
480 *****/
481
482 AdditionalDescription: "additionalDescription" Eq value = STRING;
483
484 DefaultValues: 'defaultValues' Eq value = WrapValue ;
485
486 DefaultValue:
487 (
488   'defaultValue' Eq
489   (
490     (value=STRING)
491     | (Ref ref=[AttributeValue|FQN])
492   )
493 );
494
495 OwnerPath: 'owner' Eq value = STRING ;
496
497 ClientPath: 'client' Eq value = STRING ;
498
499 UiReference:
500   'uiRef' Eq
501   (
502     ( value=STRING )
503     | (Ref ref=[AttributeValue|FQN])
```


6 Implementation

```
504 )
505 ;
506
507 DueDatePath: 'dueDateRef' Eq value=STRING ;
508
509 OutputFieldExpression: 'expression' Eq value = STRING ;
510
511 AttributeValue: name=ID Eq value = STRING | INT | FLOAT ;
512
513 // ... Other user-defined attribute definitions
514
515 TextNoQuote: /([a-zA-Z0-9-_.])*/ ;
516
517 Eq: '=' ; // Assignment Symbol
518
519 Hash: '#' ; // Directive Symbol
520
521 Ref: /(use)\s/ ;
522
523 RefStage: Ref StageTerm ;
524
525 Comment:
526   (/\s\/.*$/
527    | /\s\/.*$/
528    | /\s\/.*$/
529    | /\s\/.*$/ )
530 ;
531
532 /*****
533 ***** CP ELEMENT KEYWORDS *****
534 *****/
535
536 WorkspaceTerm: /(Workspace)\s/ ;
537
538 CaseTerm: /(Case)\s/ ;
539
540 StageTerm: /(Stage)\s/ ;
541
542 // Other keywords definitions
```

Listing 6.1: Excerpt of the Acadela grammar specifications with nonterminal and representative terminals.

6.3.3 Reflection on Addressing CP Modeling Requirements

Considering the CP modeling requirements (**F1** - **F13**), Acadela grammar addresses the constructions of medical processes (**F1**) with a Case (line 52) containing multiple Stages (line 67), each Stage (line 112) comprising Tasks (line 136) storing InputFields (line 231) that medical professionals collected during the treatment. Moreover, the InputFields and OutputFields (line 266) can be used to store data of *medical evidence* (**F3**). In this use case, the Task represents the gathering of *medical evidence*, and OutputFields hold the computed value based on the InputField(s). Another usage of InputFields is constructing checklists (**F7**) using the `#multiplechoice` directive (line 424) combined with the *question* attribute, Question and Option element (lines 238, 252, and 257). Moreover, thanks to the generic schema definition of Tasks as an object with properties, they can represent various types of information (**F8**). For example, medical documents using the `#documentLink(<UrlToDocument>)` directive (line 435) on an InputField, or dynamic data presentation (**F12**) as an OutputField based on a SVG template and rendering logic defined as InputFields. Finally, resources (e.g., medicine or medical devices) are also declarable as InputFields of an equipment allocation Task, as required by **F9**.

6 Implementation

Considering the definition of alternative workflows and their triggering conditions (**F2**), Acadela provides a Precondition element (line 199) attached to a Stages or Task as a prerequisite to activate alternative treatment processes. A Precondition defines the previous Stages or Task that must be finished (previousStep), along with an optional boolean expression that must be true to trigger the Stages or Task (condition). Furthermore, Acadela can construct *serial repetitive* or *parallel* Stages or Tasks when combining the *repeatable* directive (line 456) and Precondition, as required by **F4**.

To model the goals of treatment processes (**F5**), modelers can create a Task to collect the treatment goals in InputFields. Declaring a SummaryPanel (line 314) containing SummaryParam storing the reference path (line 323) to these InputFields displays the *treatment objective* in the patient case summary.

Assigning medical professionals to CP processes (**F6**) is possible thanks to the Responsibilities (line 73), CaseOwner (line 98), Setting Attribute (line 288) elements, and the *owner* attribute of a Stages or Task (lines 118 and 188). First, the Responsibilities section state references to existing medical groups or users in the database of a medical institution. Assigning one of these references to the CaseOwner of the Setting sets a Group or User with read and write access rights to all CP processes. Next, declaring an alias to these references as a Setting Attribute enables modelers to assign a group or medical professional to a Stages or Task under the *owner* attribute.

Modeling time constraints (**F10**) for a Task requires a combination of defining a *due date* Attribute in the Setting and declaring a reference to the Attribute as a value of the *dueDatePath* Task attribute (line 189). However, time-lapsed events are not definable in Acadela.

Communication with external systems for integration or orchestration purposes (**F11**) is possible with CaseHook, HttpHook, and DualTaskHttpHook (lines 333, 341, and 353) rules grouped under the Trigger section. These constructs enable modelers to define which type of event triggers the sending of a HTTP request to a given URL of a partner system using a HTTP method. In addition, an optional error message is declarable to provide a user-friendly bug description to medical professionals.

Finally, the Import element (line 22) helps modelers include CP elements defined from other files into the current CP meta-model (**F13**). Using the **FQN** and **FQNI** features (lines 18 and 19) of textX, Acadela also supports import by *name* and by *alias*; hence modelers can assign a namespace to a particular imported file. The syntax for *alias import* is `<alias>.<elementId>`.

When the Acadela backend receives a CP meta-model, it creates a parser based on the Acadela grammar. The parser aims to generate Python classes representing the structure of the input CP meta-model.

6.4 Parser

textX parser helps Acadela to 1) recognize *syntax errors* and 2) access *CP elements* and their *attributes* by generating a textX meta-model of the input CP definition. Acadela uses the `meta_model_from_file` to instruct textX to generate the parser and Python class meta-model from the Acadela grammar file. textX clones a new parser every time it parses an input CP definition (Dejanović, n.d.d). The parser reads the CP definition source code and creates a graph of Python classes representing CP concepts at runtime (Dejanović et al., 2017, p. 3).

Import Scoping Definition: Acadela declares a **FQNImportURI** rule from the `scoping_provider` of textX to locate a file path (URI) and import an element based on their name attribute into the current CP definition. The `scoping_provider` can have a custom converter to extract the modules declared in a file path. For example, in Acadela, a file path at the location `/folder/file1.aca` is written as `folder.file1`; the custom converter replaces the `/"` character with the `."` and removes the `".aca"` file extension. Furthermore, Acadela sets the `importAs` property to `True` to enable *alias import*. An imported object has a `ref` attribute pointing to its original Python object containing the properties parsed by textX.

Parsing the CP Definition: Because the parser's Python classes correspond to the ones of the textX meta-model, textX can convert the CP definition string into a model using the `meta_model_from_str` func-

6 Implementation

tion. The generated model is a Python object containing the meta-model, parser (`tx_parser`), first and final lines, attributes, and the CP hierarchy, i.e., A CP has a Workspace and imports, a Workspace contains Case, a Case includes Responsibilities, Setting, Stages, SummaryPanel, and CaseHook. Each element in the hierarchy is also a Python object that stores its *attributes* and *parent*. The Acadela interpreter traverses through this textX model to construct a Python class of the CP for semantic validation and compilation to SACM CP format.

The following Listings show an example of parsing a CP Definition to generate its textX model. Listing 6.2 shows the reduced code snippet of a Stage definition that imports a Hook element defined in Listing 6.3. Listing 6.4 presents the generated Python object of the *Evaluation* Stage in the textX model. Finally, Listing 6.5 and 6.6 respectively demonstrate the Python objects of the referenced Hook in the Stage and its Hook definition.

```
1 import extfile.hook
2
3 workspace Demo
4
5 define Case Hypertension
6     prefix = 'ST1'
7     // Case Attributes
8     Stage Identification // ... Stage Definition
9     Stage Evaluation
10        owner = 'Setting.CaseOwner'
11        label = 'Identification'
12        externalId = 'BpEvaluation'
13
14        Precondition previousStep = 'Identification'
15
16        Trigger use Hook hook1
17
18        HumanTask RequestMedicalTest // ... HumanTask Definition
19        HumanTask MeasureBloodPressure // ... HumanTask Definition
```

Listing 6.2: Code Snippet of a Stage Definition in a Case (Hypertension) of a Workspace (Demo)

```
1 define Hook hook1
2     On activate method Post
3     invoke 'http://partnersystem.com/activate'
```

Listing 6.3: Definition of a Hook under the file path `./extfile/hook.aca` imported in line 1 of Figure 6.2

6 Implementation

```
1 // Access by calling model.defWorkspace.case.stageList[1].__dict__
2 { '_tx_position': 2631,
3   '_tx_position_end': 4455,
4   'description': <textx:AcadelaGrammar.Description instance at 0x2006684d9b0>,
5   'directive': None,
6   // Other attributes
7   'externalId': <textx:AcadelaGrammar.ExternalId instance at 0x2006684d9e8>,
8   'hookList': [<HttpHook:>],
9   'name': 'Evaluation',
10  'ownerPath': <textx:AcadelaGrammar.OwnerPath instance at 0x2006684d550>,
11  'parent': <Case:Hypertension>,
12  'preconditionList': [<textx:AcadelaGrammar.Precondition instance at 0x2006684d320>],
13  'ref': None,
14  'taskList': [<HumanTask:RequestMedicalTest>, <HumanTask:MeasureBloodPressure>]
15 }
```

Listing 6.4: A generated Python object of the Stage Evaluation from Listing 6.2

```
1 { '_tx_position': 2864,
2   '_tx_position_end': 2878,
3   'event': None,
4   'failureMessage': '',
5   'method': None,
6   'name': '',
7   'parent': <Stage:Evaluation>,
8   'ref': <HttpHook:hook1>,
9   'url': '' }
```

Listing 6.5: A Python object of the imported Hook in the hookList attribute (line 8) of the Stage Evaluation object in Listing 6.4

```
1 { '_tx_position': 7,
2   '_tx_position_end': 106,
3   'event': 'activate',
4   'failureMessage': '',
5   'method': 'post',
6   'name': 'hook1',
7   'parent': <textx:AcadelaGrammar.Start instance at 0x1a6485aa390>,
8   'ref': None,
9   'url': 'http://partnersystem.com/activate' }
```

Listing 6.6: The Python object of the imported Hook definition when accessing the *ref* attribute at line 8 of Listing 6.5

The coming section describes how the Acadela interpreter traverses the textX model of the CP for validating the semantics, compiling the CP to SACM-compatible JSON format, and returning the compiled meta-model to the Acadela frontend for visualizing the CP.

6.5 Interpreter

From the textX model of the CP Definition code, the Acadela Interpreter performs the following operations to compile the CP into a SACM-compatible JSON format:

1. **CP Meta-model Construction:** Acadela defines each CP concept (e.g., Case, Setting, Stage, Task) as a Python class representing its schema, i.e., *attributes* and *child components*. The Acadela

6 Implementation

Interpreter traverses the textX model to extract each model item's properties. These properties store the *attributes* and *child elements* corresponding to their Python schema class. The final result is a Python object containing the CP definition, Workspace, lists of Groups, Users, Stages, Tasks, Setting Attributes, and Entities with their Attributes.

2. **Constraint Validation:** The CP definition and lists of CP elements from step one serves as the ground truth for verifying semantic and SACM-exclusive syntactic constraints applied to different CP elements. For instance, the Acadela interpreter identifies the existence of attributes defined in SACM syntax (e.g., *expression*, *uiRef*) and calls the corresponding sub-grammar to verify their syntax. Additionally, the list of Stages and Tasks helps the semantic checker quickly detect duplicated Stage or Task names without analyzing the CP object.
3. **Compilation:** If the CP object is error-free, Acadela translates each element into a JSON object conforming to the structure required by SACM. The final result is a JSON CP meta-model processible by the SACM system.

The following subsections explain the execution of each operation in detail.

6.5.1 CP Meta-model Construction

CP Meta-model Schema: The Acadela syntax relieves modelers from declaring default attribute values and calling data manipulation functions, e.g., prefixing *Case*, *Stages*, and *Tasks ID*, or converting a String InputField to a numeric type in a mathematical formula. Furthermore, Acadela must express shortened directives' values in the SACM language. These features require the Acadela backend to define the missing default or undeclared values in the CP object. Nevertheless, the modified CP definition should not overwrite the textX model because it is the ground truth of the CP structure. Therefore, Acadela defines the SACM CP meta-model from Python objects in the textX model to express CP elements. Each CP element's *attributes* and *child objects* conform to the SACM syntax and Acadela grammar. Figure 6.7 illustrates the structure of the CP meta-model.

CP Meta-model Initialization: The Acadela Interpreter defines sub-interpreters to analyze each CP element in the meta-model. When traversing the textX model, Acadela invokes the corresponding sub-interpreter to extract a CP element's *attributes* and *child elements*. For instance, when iterating the *stageList* property of a *Case* object, Acadela calls the Stage interpreter to identify the Stage's *attributes* (e.g., mandatory, label) and *child elements*, i.e., Precondition, Task, HttpHook. Then, Acadela invokes the corresponding sub-interpreters to extract the *attributes* and constructs Python objects for the *child elements* of the Stage. Finally, the Stage interpreter outputs a Stage object with Stage's attributes and *child objects* as properties. Figure 6.8 illustrates a Stage interpretation from Listing 6.4.

Automatic ID Prefixing: SACM distinguishes CaseDefinition and its elements by unique IDs, thus, modelers need to prefix the IDs of Case, Setting, Stages, and Tasks. However, Acadela relieves the users from prefixing IDs by 1) defining a *prefix* attribute in the Case and 2) prepending this prefix string to the element ID when interpreting the four types of CP elements. This automatic prefixing mechanism also applies to *reference paths* defined in the condition attribute of Preconditions.

Automatic Number Conversion: Defining a mathematical expression of OutputField requires converting any Input/OutputField of type text to number using the `number()` function. To offer convenience for modelers, Acadela automatically searches for Input/OutputFields by their ID in the expression and extracts their *type*. If the Input/OutputFields are of type *text*, *enumeration*, *longtext*, or *notype*, Acadela wraps the ID of the Input/OutputFields with the `number()` function.

Noticeably, Acadela and SACM do not round the computed value; hence the result can contain multiple decimal points. Modelers can round the result by wrapping the mathematical expression with a `round()` function.

6 Implementation

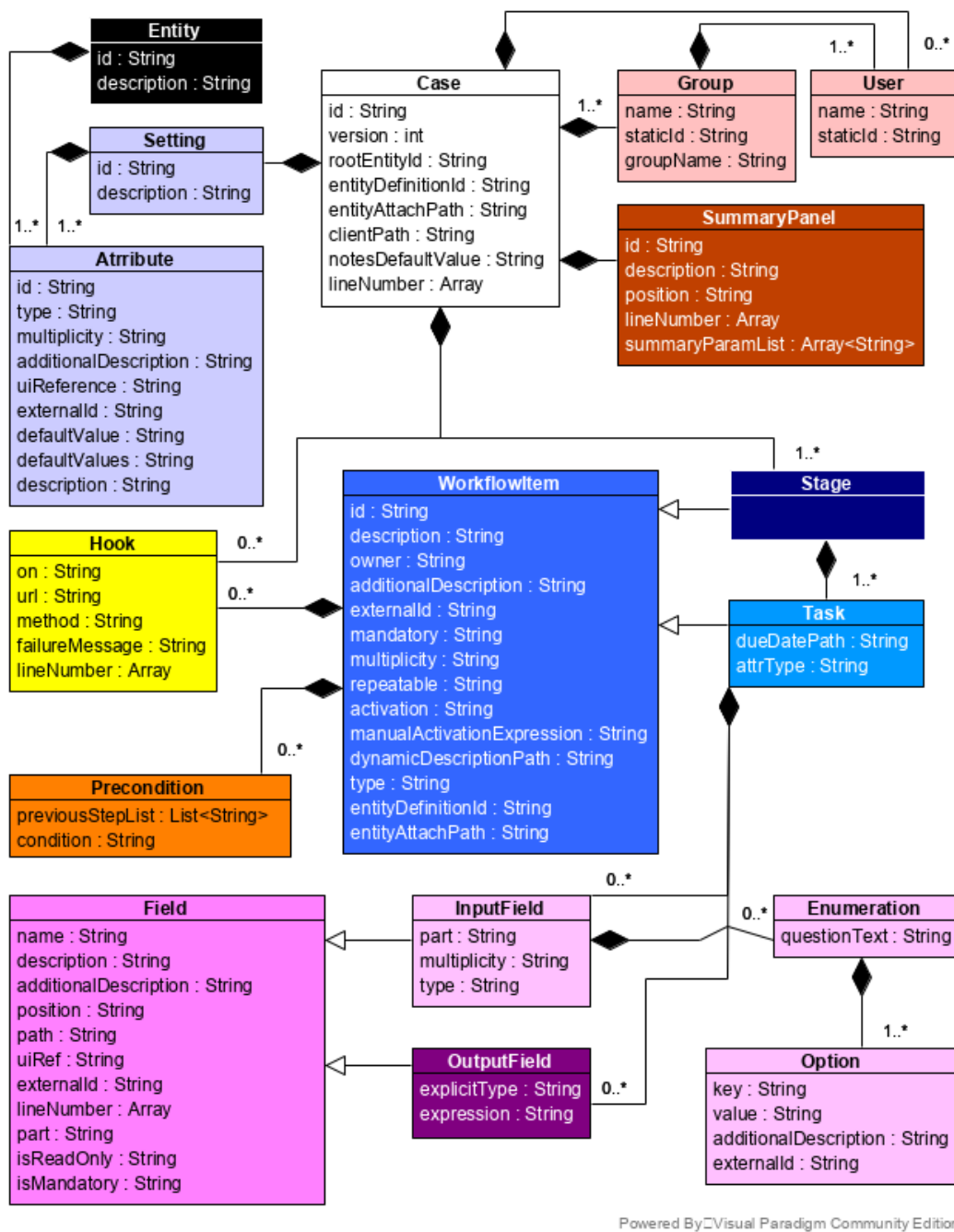


Figure 6.7: The Acadela CP meta-model based on the SACM schema and Acadela grammar. The default multiplicity is one, e.g., One Case has only one Setting.

Line Number Inclusion: The Acadela frontend has a feature that focuses the IDE on the line of a CP element definition when modelers double-click on the CP element node in the graph. Therefore, the schema of each CP element in the meta-model contains its line number in the CP element definition code. Acadela extracts this line number from the parser (`tx_parser`) of the textX model by calling the `pos_to_linecol(<cpElement>._tx_position)` function, which obtains the line of a CP element in the code from its node position in the AST.

6 Implementation

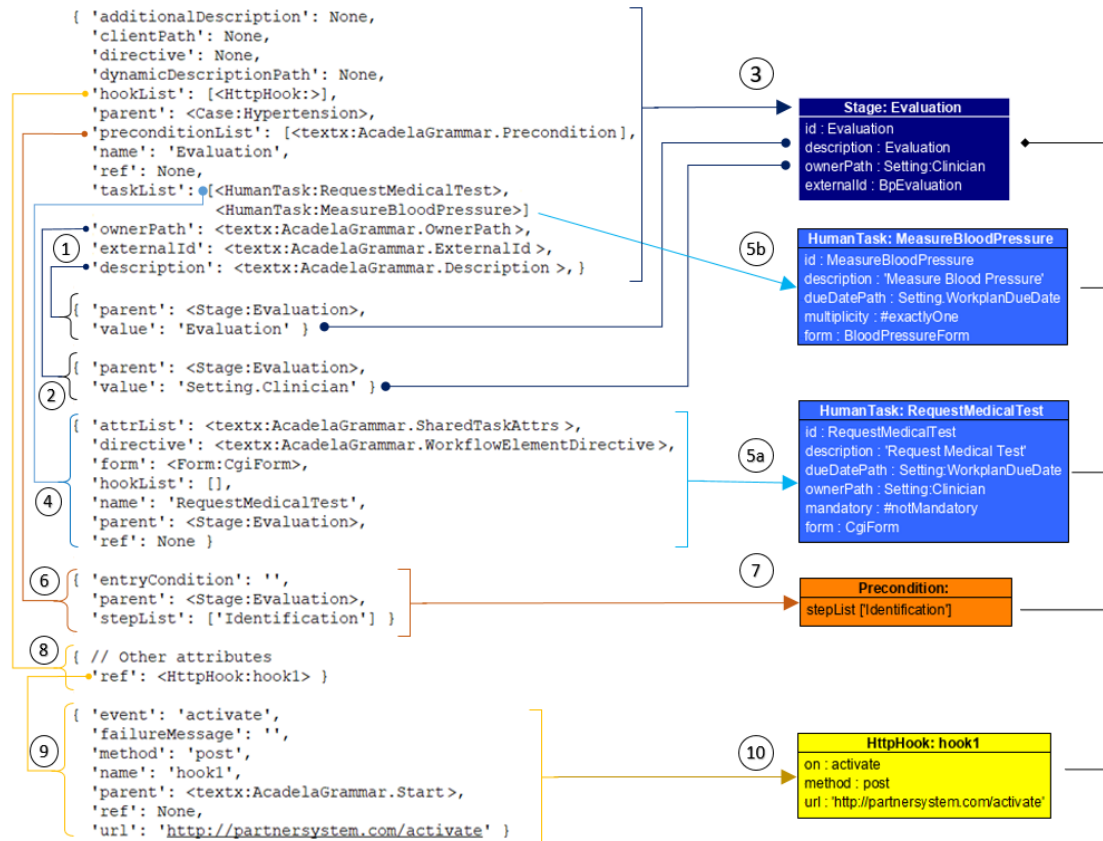


Figure 6.8: Example of how the Acadela Interpreter traverses through attributes in a Stage object of the textX model (top left) to create an Acadela Stage object with attributes and objects of child elements (HumanTasks, Precondition, and HttpHook) on the right. 1) and 2): Extract the Stage attributes from the corresponding textX model object, such as *description* (1) and *ownerPath* (2). 3) Construct a Stage object from the extracted attribute values. 4) Access each Task in the Stage to collect Task's properties. 5a) Construct a Task Object from the Task properties identified in Step 4. 5b) Construct other Tasks in the Stage by repeating Step 4. 6) Extract the previous Stage or Task ID and condition in the Precondition textX model object. 7) Construct the Precondition object based on the attributes extracted in Step 6. 8) For imported elements, e.g., HttpHook, access the reference (ref) attribute to 9) extract the attributes of the imported element. Finally, 10) Construct the HttpHook from the extracted attributes in Step 9.

Entity and Attribute Definition of Task, Stage, and Case Data: Acadela automatically creates an Entity for each Stage and Task. For Tasks, Acadela instructs the *Field interpreter* to construct each InputField as an Attribute from the *ID*, *label*, *multiplicity*, *type*, *uiReference*, *externalId*, *additionalDescription*, *defaultValue*, and *defaultValues* attributes. Additionally, each OutputField is constructed as a DerivedAttribute from the *ID*, *label*, *multiplicity*, *type*, *uiReference*, *externalId*, *additionalDescription*, and *expression* attributes. Finally, Acadela uses the Task *ID* and *label* to define a SACM Entity with the Attributes and DerivedAttribute. The Task interpreter also sets the Task type as an EntityLink to the Task Entity ID and outputs a Task Attribute using the Task *ID*, *label*, *multiplicity*, *type*, *additionalDescription*, and *externalId*.

Similarly, Acadela creates a Stage EntityDefinition from the Stage *name*, *label*, and *list* of Task AttributeDefinitions. For each Task, the Task interpreter extracts its properties to generate an AttributeDefinition of a Stage EntityDefinition.

Next, Acadela forms the EntityDefinition of the Case Setting by defining its ID and description as "Setting", and constructs AttributeDefinitions from the properties of Setting Attributes.

6 Implementation

The CaseData Entity represents the Setting and all Stages in a CP. Acadela creates this Entity with ID as *CaseData* and description as "Case Data", with the Attributes generated from the Setting and Stages attributes.

Default Attributes Definition: Acadela automatically declares typical or default attributes for the following CP elements:

1. **StaticId of Workspace, Users, and Groups:** find the *staticId* of the Workspace and each User or Group defined in the CP. Acadela calls a SACM API to get the list of Workspaces, then extract the ID of the Workspace having the same name as the declared Workspace. When interpreting the Case, Acadela calls another SACM API to collect the groups from the permission list of the Workspace and retrieve the ID of groups with the same name as the declared Groups in the CP. Similarly, Acadela pulls the *staticId* of Users by checking the matched the ID of each group's member.
2. **Owner of Stage and Task:** If a Stage has no owner, i.e., no assigned group or user, Acadela automatically assign the *CaseOwner* in the Case Setting as the Stage owner. If a Task has no owner, Acadela sets the Stage owner as the Task owner.
3. **Multiplicity, mandatory, and repeatable attributes of Stage and Task:** If not declared, Acadela sets the default value as *exactlyOne*, *true*, and *ONCE* for the three attributes. Acadela contains a configuration file to store the default values of these attributes.
4. **Mandatory and readOnly attributes of Input/OutputFields:** If undefined, set *mandatory* to **true** and *readOnly* to **false** by default.
5. **entityDefinitionId and entityAttachPath of Stage and Task:** The two attributes state the Entity-Definition ID and *path* from the root Case to the Entity, respectively. Typically, the *entityDefinitionId* of a Stage and Task is its ID. The *entityAttachPath* of a Stage is its ID since the Stage is a child of a Case; the *entityAttachPath* of the Task is `<ParentStageId>.<TaskId>` as the Task is a child of a Stage. If the two attributes are not declared, Acadela automatically assigns those typical values into the *entityDefinitionId* and *entityAttachPath* when interpreting a Stage or Task.

Order of CP Elements Interpretation: When modelers submit a CP Definition to SACM, the Acadela interprets the textX model to construct CP elements in the following order:

1. Extract the Workspace ID and its *staticId* from the root object of the textX model, which represents the Workspace definition.
2. Retrieve the Case object of the Workspace, then find the *staticId* of each Group and User defined under the Responsibilities section.
3. Access the Stage List of the Case to collect Task objects in the textX model. The Task interpreter executes the following operations:
 - a) Constructs a Python Object storing the Task attributes declared in the CP Definition
 - b) Initializes default attribute values if undefined
 - c) Calls the Field Interpreter to construct the Input/OutputFields and Attribute/DerivedAttribute objects. The Input/OutputField Interpreter validates the syntax of uiReference and OutputField's expression (See 6.5.2 in 6.5.2)
 - d) Creates the Task as an Entity and as an Attribute.

In step c), the Field interpreter takes the Input/OutputField object in the Form of a Task; the path to the Input/OutputField from the Stage, i.e., `<StageId>.<TaskId>.<FieldId>`; global Form directive(s) definition; and the textX model of the CP definition as parameters. The last parameter helps Acadela get each Field's *line number* and declares it as an Input/OutputField attribute. The other parameters enable Acadela to define declared or default attributes.

6 Implementation

For example, Acadela sets the Form directives to all Input/OutputFields, except the ones that specify their directive value. For instance, defining a global `#readOnly` directive in the Form will set all Input/OutputFields' `readOnly` attribute as `true`. However, if a Input/OutputField has a `#notReadOnly` directive, then its `readOnly` value is `false`.

4. Read the Stage object in the textX model to extract its attributes. Then construct a Stage object, a Stage as Entity, and a Stage as Attribute. Acadela uses the latter as the Attribute of the CaseData.
5. Interpret the Setting of the Case by creating a Setting Entity containing the Setting Attributes. Acadela also constructs an Attribute object from the Setting to include it in the CaseData Entity.
6. Create a CaseDefinition with `CaseOwner` and references to the patient list (CasePatient) declared in the Setting. Set the prefixed Setting as the `entityDefinitionId` and `entityAttachPath` of the CaseDefinition.
7. Construct the CaseData Entity from Attributes of the Setting and Stages.
8. Interpret Case's HttpHooks and SummarySectionDefinition from the corresponding textX model.
9. Generate a Case Object containing the Workspace and echFont(CaseDefinition) objects, along with seven separate lists of Users, Groups, Entities, Attributes, Setting attributes, Stages, and Tasks. This object serves as a cache for the syntax and semantic error analyzers to retrieve necessary data without traversing the Case object.

After interpreting the CP definition as a Case Object, Acadela validates the semantic constraints of the CP elements.

6.5.2 Syntax Error Validation

Syntax Analyzer of the textX Parser: The textX meta-model contains the syntax rules (Dejanović et al., 2017, p. 4), to verify if the model infringes any syntactic constraints and consequently throws a `textXSyntaxErrorException`. By catching the exception during parsing, Acadela extracts the violated `rule name`, `surrounding code snippet`, `line`, and `column` of the error in the `textXSyntaxErrorException`. Additionally, the Acadela syntax analyzer stores an *enhanced description dictionary* as a key-value pair data structure, with violated rule names as keywords and their explanation as values. Appendix A.10 presents the content of the *enhanced description dictionary*.

Furthermore, Acadela extracts the lists of directives and attributes keywords from a dedicated rule defined in the grammar. These keywords serve as the dictionary for detecting typos by comparing the distance of the erroneous text with every keyword.

When a `textXSyntaxErrorException` occurs, Acadela reformats the error message as follows:

1. Check if the violated rule name is a missing **Equal sign** (Eq), **STRING**, or **INT** type. The **STRING** rule indicates that the syntax error occurs when a text is not wrapped with a pair of single or double quotes. Meanwhile, the **INT** rule suggests that Acadela expects a number at the error location. Acadela replaces the rule name with the corresponding human-readable messages in these three cases and prepends the "Expected" word. For example, replacing 'STRING' with 'Text with quotation marks ("", ")'. The final error cause is **Expected Text with quotation marks ("", ")**.
2. Verify if a **typo** causes the syntax error. A misspelled word is the word at the column number of the error string. The Acadela syntax analyzer extracts the misspelled word and uses the `correction` function to return the keyword in the dictionary with the *Levenshtein distance* of less than three (Norvig, 2022) compared to the misspelled word. This *Levenshtein distance* constraint means Acadela can suggest the correct keyword if the *typo* has at most **two characters different** from any word in the dictionary. If no such keyword exists, the `correction` function returns the *typo*, and the error message only displays the *typo* without any suggested keyword.
3. For other errors, replace the violated rule name with a user-friendly explanation from the *enhanced description dictionary*, and remove redundant whitespaces.

6 Implementation

Finally, Acadela crafts an custom syntax error message with its enhanced description of the error cause. If a *typo* is detected, this message includes the potentially correct Acadela keyword. Additionally, the syntax error handler inserts the line and column numbers and the erroneous code snippet extracted from the `textXSyntaxErrorException`.

Custom Syntax Error Analyzer: To reduce the length and complexity of the base grammar after interpreting an Input/OutputField. Acadela verifies the syntax of *uiReference* and *expression* attributes which must conform to an exclusive syntactic rule of SACM. The *uiReference* can contain the color band applied to a numeric value by calling the SACM colors function. For example, `colors(5 <= red <= 10 < green < 15)` means a red background color is applied to the Input/OutputField if its value is from 5 to 10. Otherwise, SACM paints a green background if the Field value is larger than 10 and less than 15. Listing 6.7 defines the sub-grammar to verify the *uiReference* value.

```
1 UiRef :
2   ColorCodeDef | 'privatelink' | 'hidden' | 'svg' | 'linediagram'
3 ;
4
5 ColorCodeDef :
6   'colors' '(' (CompareExpression) ')'
7 ;
8
9 CompareExpression :
10  NUMBER (Comparator ColorName Comparator NUMBER)+
11 ;
12
13 ColorName :
14   'red' | 'blue' | 'green' | 'orange' | 'yellow'
15 ;
16
17 Comparator :
18   '<=' | '<>' | '<=' | '>=' | '<' | '>'
19 ;
```

Listing 6.7: Sub-grammar to verify the *uiReference* value in SACM

The grammar states that a color band function rule (`ColorCodeDef`), link (*privatelink*), *hidden*, graphical template (*svg*), and *linediagram* are legitimate values of the *uiRef*. The `ColorCodeDef` rule starts with the `colors` function that accepts a `CompareExpression` that begins with a number, followed by at least one `Comparator ColorName Comparator` number sequence. A valid `Comparator` is one of the six comparison symbols, while an acceptable `ColorName` is *red*, *blue*, *green*, *orange*, and *yellow*.

Validating a conditional statement in the expression attribute follows a similar approach. The syntax of the conditional statement in SACM is as follows:

```
1 if (<booleanExpression >) then <outputValue1 >
2 else if (<booleanExpression >) then <outputValue2 >
3 else <outputValue3 >
```

SACM accepts zero or multiple else-if clauses. The parentheses wrapped around a boolean expression are compulsory. Listing 6.8 presents the grammar used to verify the syntax of conditional statements.

```
1 IfElseStatement :
2   ifStatement elseIfStatement* elseStatement
3 ;
4
5 ifStatement :
6   /(if)\s/ conditionalExpr thenStatement
7 ;
8
```

6 Implementation

```
9  elifStatement:
10  /(else\sif)\s/ conditionalExpr thenStatement
11  ;
12
13  elseStatement:
14  /(else)\s/ STRING
15  ;
16
17  conditionalExpr:
18  (compoundStatement | complexStatement)
19  ;
20
21  complexStatement:
22  '(' compoundStatement (andOr compoundStatement)* ')',
23  ;
24
25  compoundStatement:
26  '(' Condition (andOr Condition)* ')',
27  ;
28
29  Condition:
30  TextNoQuote Comparator ( NUMBER | STRING )
31  ;
32
33  Comparator:
34  '<=' | '<>' | '<=' | '>=' | '<' | '>',
35  ;
36
37  TextNoQuote: /[a-zA-Z]*/ ;
38
39  andOr: /(and)\s/ | /(or)\s/ ;
40
41  thenStatement: "then" STRING ;
```

Listing 6.8: Sub-grammar to verify the conditional statement of expression attribute in SACM

The first four rules from lines 1 to 15 define the structure of *if*, *else if*, and *else* clauses in the conditional statement. The next three rules from lines 17 to 27 enforce the structure of *compound* and *complex conditional statements*. A *compound statement* consists of two boolean expressions connected by the *and* or *or* operator. A *complex statement* consists of a boolean expression and *compound statements* grouped by parentheses. The *Condition* rule declares the syntax of a boolean expression, which accepts text on the LHS, a *Comparator*, and a number or string on the RHS, e.g., `Field1 < 5`.

If a syntax error occurs, the syntax analyzer applies the *enhanced description dictionary* (See *Syntax Error Validation* in Section 6.4) to display text from regular expressions in the grammar. For example, in the returned error message, replace the rule `(if\s)` of the grammar by the *if* word.

6.5.3 Semantic Error Validation

ID Uniqueness After interpreting the CP, Acadela stores lists of Groups, Users, Setting Attributes, Stages, and Tasks. The Acadela semantic analyzer traverses these lists to verify that their elements' ID satisfies the following constraints:

1. Each Group, User definition has a unique ID.
2. No two Attributes in the Setting have the same ID. Attribute ID is not the same as any Stage ID.
3. Each Stage ID is unique among other Stages and Tasks. The semantic analyzer examines IDs in the two lists of Stages and Tasks for this detection.

6 Implementation

4. Each Task ID is unique among Tasks in the same Stage. Acadela extracts the list of Tasks in every Stage object and checks that no two Tasks have the same ID.
5. Each Input/OutputField ID is unique among others in the same Task. The semantic analyzer iterates through each Task and extracts its Input/OutputFields, then verifies that no ID is defined twice among the Input/OutputFields.

Violating any of the above constraints triggers Acadela to generate an error message stating:

"[duplicated element type] ID should be unique! [duplicated ID] at line [line number] and column [column number] is a duplicate. Please verify that the IDs are unique for each [duplicated element type]."

For example, if two Stages share the same ID Evaluation, and the position of the first Evaluation Stage is line 54 and column 5, then the error message is:

"Stage IDs should be unique! Evaluation at line 54 and column 5 is a duplicate. Please verify that the IDs are unique for each Stage."

If a Task ID is the same as a Stage, the error message has the following pattern:

Task IDs should be unique! [Task Name] at line [line number] and column [column number] is duplicated. Please verify that the Task ID does not match with a Stage ID.")

Valid Reference Path The Acadela semantic analyzer ensures that *references* declared in any CP element point to an existing object by performing the following verifications:

1. For each Task and Stage, the *dueDateRef* and *owner* attributes must point to the corresponding Setting Attributes. The *dueDateRef* and *owner* values are typically `Setting.<AttributeID>`. Acadela splits the text to extract the `<AttributeID>` part and searches in the list of Setting Attributes for elements having the same ID.

For the *owner* attribute, Acadela further checks if the Setting Attribute links to an existing User or Group in the Responsibilities section. Meanwhile, the *dueDatePath* reference must point to an Attribute of type `date`.

2. For each Precondition in a Stage or Task, each ID defined in the *previousStep* list must match an existing Task or Stage ID. Acadela extracts every element in the *previousStep* attribute and searches in the lists of Stage IDs and Task IDs.

Furthermore, each *reference* in the boolean expression of the *condition* attribute must point to an existing Setting Attribute or Input/OutputField. To verify this constraint, the semantic analyzer extracts *reference paths* in the *condition* attribute value. Acadela uses the Python regular expression to split and filter the *references* from other characters, i.e., *comparators*, *and*, *or*, *textit+*. The final result is a list of paths that conform to either the `Setting.<AttributeID>` or `<StageID>.<TaskID>.<FieldID>` pattern.

For each *reference path*, if it starts with "Setting", Acadela extracts the second part and searches for a matched Attribute ID. Otherwise, it is a *reference* pointing to an Input/OutputField. Thus, Acadela checks if the first part is in the list of Stages, the second part is the same with any Task ID of the Stage, and the third path corresponds to an Input/OutputField of the matched Task.

3. For the custom ElementPath) in an Input/OutputField and InfoPaths in SummaryPanels, Acadela uses the path validation mechanism described in the previous paragraph to verify the *references*.
4. If an OutputField's *expression* attribute contains a conditional statement, Acadela uses regular expressions to extract Input/OutputFields ID inside boolean expressions. Afterward, the semantic analyzer checks if each extracted ID refers to an Input/OutputField in the same Task Form.

6 Implementation

Regarding the generated error message content, considering the *owner* and *dueDateRef* attributes, Acadela crafts a semantic error message containing the line number and unfound Setting Attribute ID as follows:

Semantic Error at line [line number]! Owner [Attribute ID] not found in Settings.

Furthermore, if the owner Setting Attribute links to a non-existing Group or User, the error message states the line number and the Group ID or User ID not found in the declared Groups or Users.

Semantic Error at line [line number]! CaseOwner '[Group ID]' not found in groups.

Turning to the Precondition element, if any *step* element in the *previousStep* does not match with any Stage or Task ID, then the error message is as follows:

Semantic Error at line [line number]! Task or Stage '[step]' in precondition not found.

Regarding the reference path, if Acadela detects the phrase "Setting" in the first part but the second part does not match any Setting Attribute ID, then the error message states the erroneous ID and reminds the user of the correct reference pattern:

Semantic Error at line [line number]!
Invalid precondition path '[reference path]'. The path does not point to an existing element. Make sure your path follows the below rules:

Setting.<AttributeName>

Otherwise, Acadela expects a *pattern* pointing to an Input/OutputField. Therefore, if the pattern does not have three parts, the outputted error is:

Semantic Error at line [line number]!
Invalid precondition path '[reference path]'. The path does not point to an existing element. Make sure your path follows one of these rules:

1.<StageName>.<TaskName>.<FieldName>
2.<Setting>.<AttributeName>

If any part of the Input/OutputField reference path is incorrect, Acadela includes the faulty part in the error message. For example, including a Task ID that does not correspond to any child Task of any Stage results in the below error message:

Semantic Error: Invalid reference path at line [line number]! '[Declared Task ID]' Task does not exist. Expect the ID of a defined Task in the Case.

If an Input/OutputField ID declared in the expression attribute of an OutputField does not exist, Acadela constructs the following error message:

Semantic Error at line [line number]! Invalid field [Unfound Input/OutputField ID] found in the expression of OutputField [expression's OutputField ID]. Expected the ID of an InputField or OutputField declared in the same Form as OutputField [expression's OutputField ID]."

6 Implementation

Trusted API Validation Acadela persists a table of trusted API URLs and their legitimate HTTP methods for each `Workspace` (i.e., medical institution). An e-Health system can store this table inside a database or a file. In the current implementation, Acadela persists the trusted API tables as a CSV file, as shown in Table 6.1.

Workspace	URL	Allowed HttpMethod
StPaul	http://connecare.de:3001/connecare	"POST"
StPaul	http://partnersystem.de/record/bloodpressure	"POST"
Demo	https://externalsystem.com/monitor	"POST , GET"

Table 6.1: Example of a trusted API table as a CSV file. Each row stores the `Workspace` ID, the API URL of an external system, and its eligible HTTP method(s).

The (`Workspace`) column refers to the `Workspace` ID declared at the beginning of the CP Definition. The `Workspace` represents medical institutions that require external communications. The `URL` column represents the APIs the partner systems provide to the medical institution. Modelers need to declare the complete URL of each communicable API endpoint. Finally, the `HttpMethod` column represents the HTTP method that the partner system authorizes the medical institution to use. For example, if the permitted HTTP method to a URL is `GET`, then the e-Health system can not request the URL with a `DELETE` HTTP method. This feature prevents calling unauthorized services from an e-Health system.

HttpHook Validation: Acadela applies the trusted API table to verify each `HttpHook` object of any `Case`, `Stage`, or `Task`. For each `HttpHook`, the semantic analyzer checks if the CP `Workspace` and `HttpHook` URL match the `Workspace` and `URL` value, respectively. Finally, the analyzer verifies whether the `HttpHook` method is in the list of permitted methods.

Generate Error Messages: Acadela crafts an error message stating that the declared URL is not in the list of trusted URLs for a `Workspace` if the semantic analyzer cannot find any matched URL for a given `Workspace` in the table. The structure of the error message is as follows:

The URL [HttpHook URL] at line [line number] is not in the list of trusted sources for the workspace [CP's Workspace]. Please check the trusted sources list for the permitted URLs.

If the `HttpHook` URL is among the trusted URLs of a `Workspace`, but the declared `HttpHook` method does not apply to the URL, then Acadela generates the following error message:

The URL [HttpHook URL] at line [line number] does not accept the HTTP method [declared HTTP method]. Allowed methods: [trusted HTTP methods]. Please further check the trusted sources list for the permitted methods.

After transforming the input CP definition to a Python object, Acadela starts the compilation process to represent the CP object in the SACM JSON structure.

6.6 Compilation to SACM Clinical Pathway

The objective of the compilation step is to create a CP meta-model compatible with SACM from the CP Definition Python Object produced by the interpreter. Thanks to the CP meta-model, SACM can generate CP models for treating patients according to the guidelines of the CP.

Acadela compiles the *attributes* and *child elements* of each CP concept in the CP Definition object into the JSON structure required by SACM. Listing 6.9 provides an overview of the SACM JSON structure for representing the CP Definition.

6 Implementation

```
1 {
2   "jsonTemplate": {
3     "SACMDefinition": {
4       "Workspace": [
5         {
6           "$": {
7             "staticId": "<WorkspaceStaticId>",
8             "id": "<WorkspaceId>"
9           },
10          "EntityDefinition": [
11            {
12              "$": {
13                "id": "<prefix>_CaseData",
14                "description": "Case Data"
15              },
16              "AttributeDefinition": [
17                { "$": { <Stage1 properties> } }},
18                { "$": { <StageN properties> } }},
19                { "$": { <Setting properties> } } }
20            ]
21          },
22          {
23            "$": {
24              "id": "<prefix>_Setting",
25              "description": "Setting"
26            },
27            "AttributeDefinition": [
28              { "$": { <Case Owner properties> } }},
29              { "$": { <Setting Attribute1 properties> } }},
30              { "$": { <Setting AttributeN properties> } } }
31            ]
32          },
33          {
34            "$": {
35              "id": "<prefix>_Stage1Id",
36              "description": "<Stage1Description>"
37            },
38            "AttributeDefinition": [
39              {
40                "$": { <Stage1 Task1 properties> }},
41                "$": { <Stage1 TaskN properties> } }
42            ]
43          },
44          {
45            "$": {
46              "id": "<prefix>_Stage1Task1Id",
47              "description": "Stage1Task1Description"
48            },
49            "AttributeDefinition": [
50              {
51                "$": { <Stage1Task1InputField1 properties> }},
52                "$": { <Stage1Task1InputFieldN properties> } }
53            ]
54          },
55          "DerivedAttributeDefinition": [
56            {
57              "$": { <Stage1Task1OutputField1 properties> }},
58              "$": { <Stage1Task1OutputFieldN properties> } }
59            ]
60          },
61          {
62            "$": {
63              "id": "<prefix>_Stage1TaskNId",
64              "description": "Stage1TaskNDescription"
65            },
66            "AttributeDefinition": [
67              {
68                "$": { <Stage1TaskNInputField1 properties> }},
69                "$": { <Stage1TaskNInputFieldN properties> } }
70            ]
71          }
72        ]
73      }
74    }
75  }
76 }
```

6 Implementation

```

70         "$": { <Stage1TaskNInputField1 properties> },
71         "$": { <Stage1TaskNInputFieldN properties> }
72     }
73 ],
74     "DerivedAttributeDefinition": [
75     {
76         "$": { <Stage1TaskNOutputField1 properties> },
77         "$": { <Stage1TaskNOutputFieldN properties> }
78     }
79 ],
80 },
81 { <Entity Definition of StageN as lines 33-44> },
82 { <Entity Definitions of StageN's Tasks as lines 45-80> },
83 ],
84 "CaseDefinition": [
85 {
86     "$": { <CP properties and Case HttpHook definitions> },
87     "SummarySectionDefinition": [
88     {
89         "$": { <SummarySection1 properties> },
90         "SummaryParamDefinition": [
91         { "$": { "path": "<ReferencePath.To.Input/OutputField1>" } }
92         { "$": { "path": "<ReferencePath.To.Input/OutputFieldN>" } }
93         ]
94     },
95     {
96         "$": { <SummarySectionN properties> },
97         "SummaryParamDefinition": [
98         { "$": { "path": "<ReferencePath.To.Input/OutputFieldY>" } }
99         { "$": { "path": "<ReferencePath.To.Input/OutputFieldZ>" } }
100        ]
101    }
102    ]
103 }
104 ],
105 "StageDefinition": [
106 {
107     "$": { <Stage1 properties> },
108     "SentryDefinition": [
109     {
110         "precondition": [
111         { "$": { <Precondition1 PreviousStep1 and Condition> } } },
112         { "$": { <Precondition1 PreviousStepN and Condition> } } }
113     ]
114     },
115     {
116         "precondition": [
117         { "$": { <PreconditionN PreviousStep1 and Condition> } } },
118         { "$": { <PreconditionN PreviousStepK and Condition> } } }
119     ]
120     }
121 ],
122 "HttpHookDefinition": [
123 { "$": { <Stage1 HttpHook1 properties> } } },
124 { "$": { <Stage1 HttpHookN properties> } } }
125 ],
126 "$$: [
127 {
128     "$": { <Task1 properties> },
129     "#name": "<TaskType>",
130     "SentryDefinition": [
131     // Same declaration structure as lines 109-120
132     ],
133     "HttpHookDefinition": [
134     { "$": { <Task1 HttpHook1 properties> } } },
135     { "$": { <Task1 HttpHookN properties> } } }
136 ],
137     "TaskParamDefinition": [
138     { "$": { <Task1 Input/OutputField1 properties> } } },
139     { "$": { <Task1 Input/OutputFieldN properties> } } },

```


6 Implementation

```
140         ]
141     },
142     { // Other Task declarations follows the structure of lines 127
143         -141 }
144     ]
145 }
146 ]
147 },
148 ],
149 "Group": [
150     {
151         "$": {
152             "staticId": "<Group1StaticId>",
153             "id": "Group1Name"
154         }
155     },
156     {
157         "$": {
158             "staticId": "<GroupNStaticId>",
159             "id": "GroupNName"
160         }
161     }
162 ],
163 "User": [
164     {
165         "$": {
166             "staticId": "<User1StaticId>",
167             "id": "<User1Id>"
168         }
169     },
170     {
171         "$": {
172             "staticId": "<UserNStaticId>",
173             "id": "<UserNId>"
174         }
175     }
176 ]
177 ]} // End of Workspace Scope
178 }
179 }
```

Listing 6.9: Structure of a CP Definition in SACM JSON format. Variables are wrapped inside < and > symbols.

The SACM CP Definition stores the *attribute* of a CP element in the \$ object. The SACM CP Definition starts with a Workspace containing its properties, a CaseDefinition, and three lists of EntityDefinitions, Groups, and Users. The EntityDefinitions list stores the CP structure (i.e., Setting and Stages); CP Setting; Stages; and Tasks as SACM EntityDefinitions. Every EntityDefinition has its AttributeDefinitions; The AttributeDefinitions of a Setting EntityDefinition are Setting Attributes; A Stage EntityDefinition has Tasks as AttributeDefinitions; a Task EntityDefinition has InputFields as AttributeDefinitions and OutputFields as DerivedAttributeDefinitions. Each Attribute/DerivedAttributeDefinition stores the properties of its object.

Turning to the CaseDefinition, it stores the CP properties, SummarySectionDefinitions, and StageDefinitions. Each SummarySectionDefinition stores SummarySections with their properties and path(s) to CP elements. Meanwhile, the StageDefinition list stores the Stage properties and SentryDefinitions (Preconditions), HttpHookDefinitions, and TaskDefinitions of every CP Stage. Each TaskDefinition contains the Task properties, HttpHookDefinitions, SentryDefinitions, and TaskParamDefinitions. Each TaskParamDefinition holds an Input/OutputField *mandatory*, *readOnly*, and *path* properties. The latter is the reference to the Input/OutputField from its parent Stage and Task.

Finally, the Workspace stores the Groups, each with a group *name* and *staticId* as properties. Similarly, the properties of each User in the Users list are an internal *ID* and the global *staticId*.

6 Implementation

In addition, to support the feature of focusing on a code when double-clicking a CP object in the visualization, Acadela stores the line and row number as a property in each StageDefinition, TaskDefinition, HttpHookDefinition, SentryDefinition, and TaskParamDefinition (InputField and OutputField).

To compile the CP conforming to the above structure, Acadela traverses through the CP Definition Python Object to construct JSON objects for each CP element from their properties and child items. The step of traversal is as follows:

1. **Compile the CaseDefinition:** Acadela creates a JSON object from the Case object to represent the CaseDefinition with attributes in the below format:

```
1 {
2   "\$": {},
3   "SummarySectionDefinition": [],
4   "StageDefinition": []
5 }
```

Acadela constructs key-value pairs from the Case properties, with the key as the property name and the value as the property data. These properties include the Case HttpHook definitions. Figure 6.9 illustrates an example of constructing the Case properties as the \$ JSON object (right) from the interpreted Case Python Object (left).

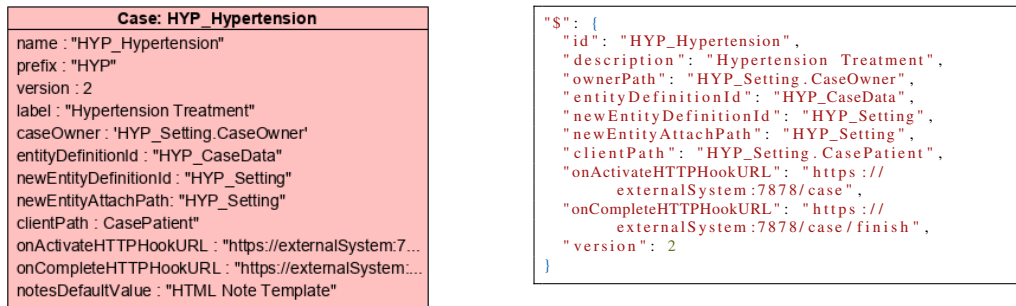


Figure 6.9: Example of compiling properties of an interpreted Case object (left) into SACM JSON format (right).

2. **Insert SummarySectionDefinitions:** Acadela accesses each Summary in the SummarySection list of the Case to extract the Summary properties and its SummaryParam, which stores the list of reference paths to CP elements. Finally, Acadela constructs each SummarySection into an SACM JSON structure, as illustrated by Figure 6.10.
3. **Compile StageDefinitions:** For each Stage in the interpreted Stages list, Acadela constructs the following JSON elements:
4. **Compile TaskDefinitions for each StageDefinition:** Acadela retrieves the list of Task objects in each Stage. Next, Acadela creates a TaskDefinition JSON object to store the *properties* and *child elements* in each interpreted Task. Specifically, the TaskDefinition contains a "\$" JSON object representing Task properties as key-value pairs. The TaskDefinition has a #name key to indicate the Task type, i.e., HumanTask, AutomatedTask, or DualTask. Like StageDefinitions, Acadela expresses HttpHooks and Preconditions in the Task as HttpHookDefinition and SentryDefinition arrays. Finally, a TaskParamDefinition array stores the properties of Input/OutputFields of the Task.

6 Implementation

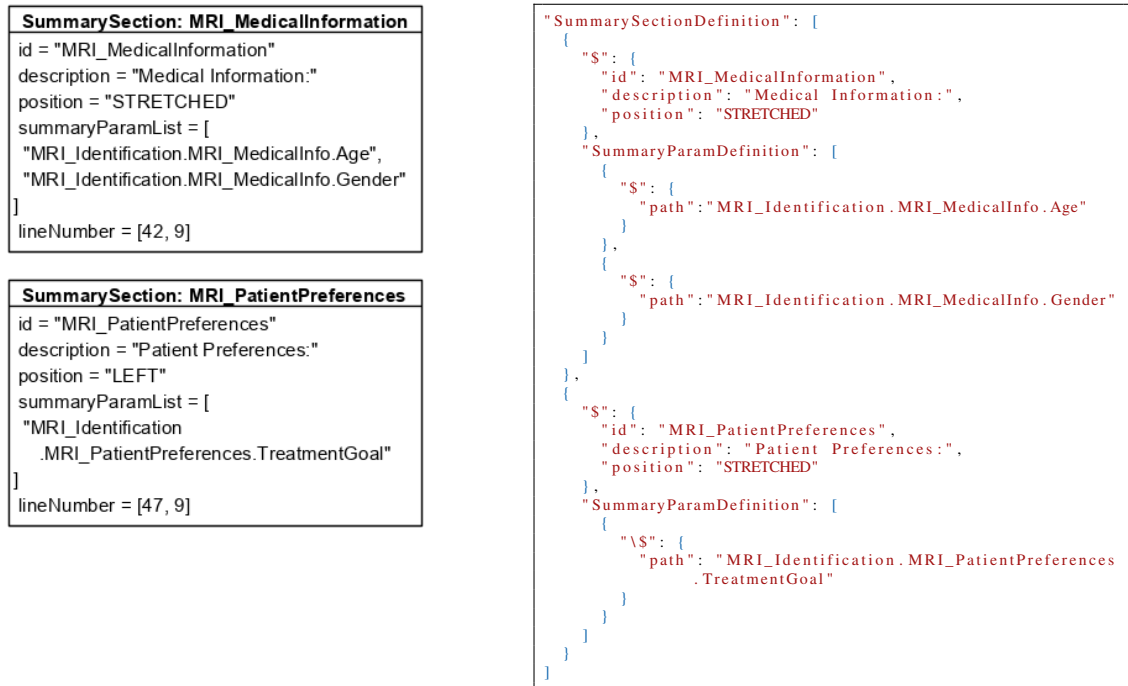


Figure 6.10: Example of compiling an interpreted SummarySection object (left) into SACM JSON format (right). The SummarySection code snippet is from lines 42 to 49 of Listing 5.23.

5. **Compile Input/OutputFields as TaskParamDefinitions:** Acadela retrieves the list of Input/OutputFields in a Task and extracts their ID, *path*, *readOnly*, *mandatory*, *position*, *line* and *row* numbers, *Field type* (InputField or OutputField) properties. The *Field type* helps the Acadela IDE to choose the appropriate color for painting the InputField or OutputField box. Meanwhile, the ID saves the IDE effort in extracting the Input/OutputField ID from the SACM path value. Figure 6.11 shows an example of constructing a *MeasureBmi* HumanTask with two InputFields (Height and Weight) and one OutputField.

Next, Acadela generates a "\$" JSON object to store these properties. The compilation of Input/OutputFields for all Tasks completes the construction of the StageDefinition array.

- a) A "\$" JSON object of the Stage properties.
- b) A HttpHookDefinition JSON array with each element containing the properties of HttpHooks in the Stage.
- c) A SentryDefinition JSON array storing Precondition definitions, each containing the ID of Stage or Task as previousStep, and the transition condition.
- d) The child Tasks array wrapped under a "\$\$" object, which stores an array of TaskDefinitions.

Figure 6.12 demonstrates an example of compiling an interpreted Stage object into SACM JSON structure.

6 Implementation

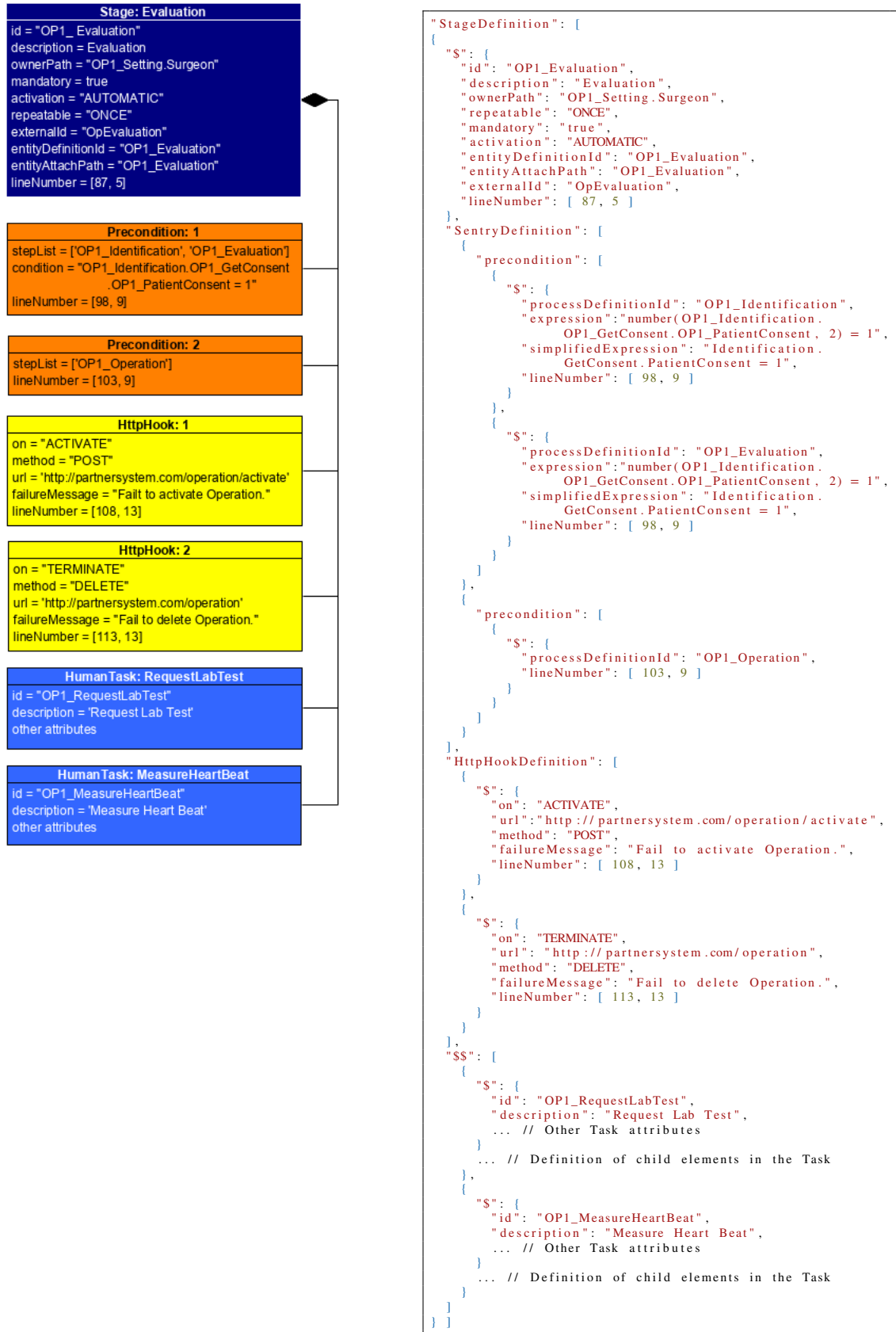


Figure 6.12: Example of compiling properties of an interpreted Stage object (left) into SACMJSON format (right).

6 Implementation

6. **Compile Workspace properties:** Construct a Workspace object with a "\$" object storing the ID and *staticId* of the interpreted Workspace. Additionally, the Workspace contains EntityDefinition and CaseDefinition array. The array contains a CaseDefinition with Case properties and CP elements produced from steps 1 to 5. The JSON object of the Workspace at this step is as follows:

```

"Workspace": [
  "$": {
    "staticId": "<WorkspaceStaticId>",
    "id": "<WorkspaceId>"
  },
  "EntityDefinition": [],
  "CaseDefinition": [ {
    <Case properties and CP elements generated from steps 1 to 5>
  } ]
]

```

7. **Compile EntityDefinitions and their AttributeDefinitions:** From the list of interpreted Entities, Acadela constructs the corresponding EntityDefinition comprises a "\$" object storing the Entity properties and an array of AttributeDefinitions generated from the Entity's Attribute. Each AttributeDefinition contains a "\$" object holding the Attribute properties. Section 6.5.1 describes the creation of EntityDefinitions and AttributeDefinitions of the CaseData, Setting, Stages, and Tasks. Since the generation of CaseData, Setting, and Stages follows the same mechanic, Figure 6.13 illustrates the EntityDefinition compilation of a Stage, which is similar to compiling the ones of Setting and CaseData.



Figure 6.13: Example of compiling a interpreted Stage (left) into SACM JSON structure of EntityDefinition (right). The Attribute-related properties of HumanTasks in the Stage constitute their AttributeDefinitions.

6 Implementation

If the EntityDefinition specifies the Schema of a Task, Acadela creates a DerivedAttributeDefinitions array to store dedicated DerivedAttribute properties of the Task's OutputFields. Meanwhile, the AttributeDefinition expresses the properties of InputFields in the Task. For *single-* or *multiple-choice* InputFields, Acadela compiles an EnumerationOption JSON array to store the options, as illustrated in Figure 6.14.

8. **Compile Groups and Users:** Generates two arrays of Group and Users declared in the CP. Each Group or User JSON object contains a "\$" object storing their ID and *staticId*. The completion of the Users and Groups construction marks the finish of the Workspace compilation.
9. **Construct a SACM CP Definition:** Creates a SACMDefinition object storing the complete Workspace object. Acadela wraps the SACMDefinition inside a jsonTemplate JSON object. When SACM receives the jsonTemplate, it directly process the CP meta-model sent from Acadela.

Finally, Acadela sends the final SACM CP Definition JSON object as a POST request to the SACM CP generation API. Upon receiving the request, SACM analyzes and persists the CP elements. If any error occurs, SACM returns an error response to the Acadela backend, which notifies the Acadela IDE that an internal error occurs.

A successful compilation or validation returns the SACM CP Definition JSON Object to the Acadela IDE, which renders a "successfully compiled" message and constructs the CP visualization accordingly.

6 Implementation

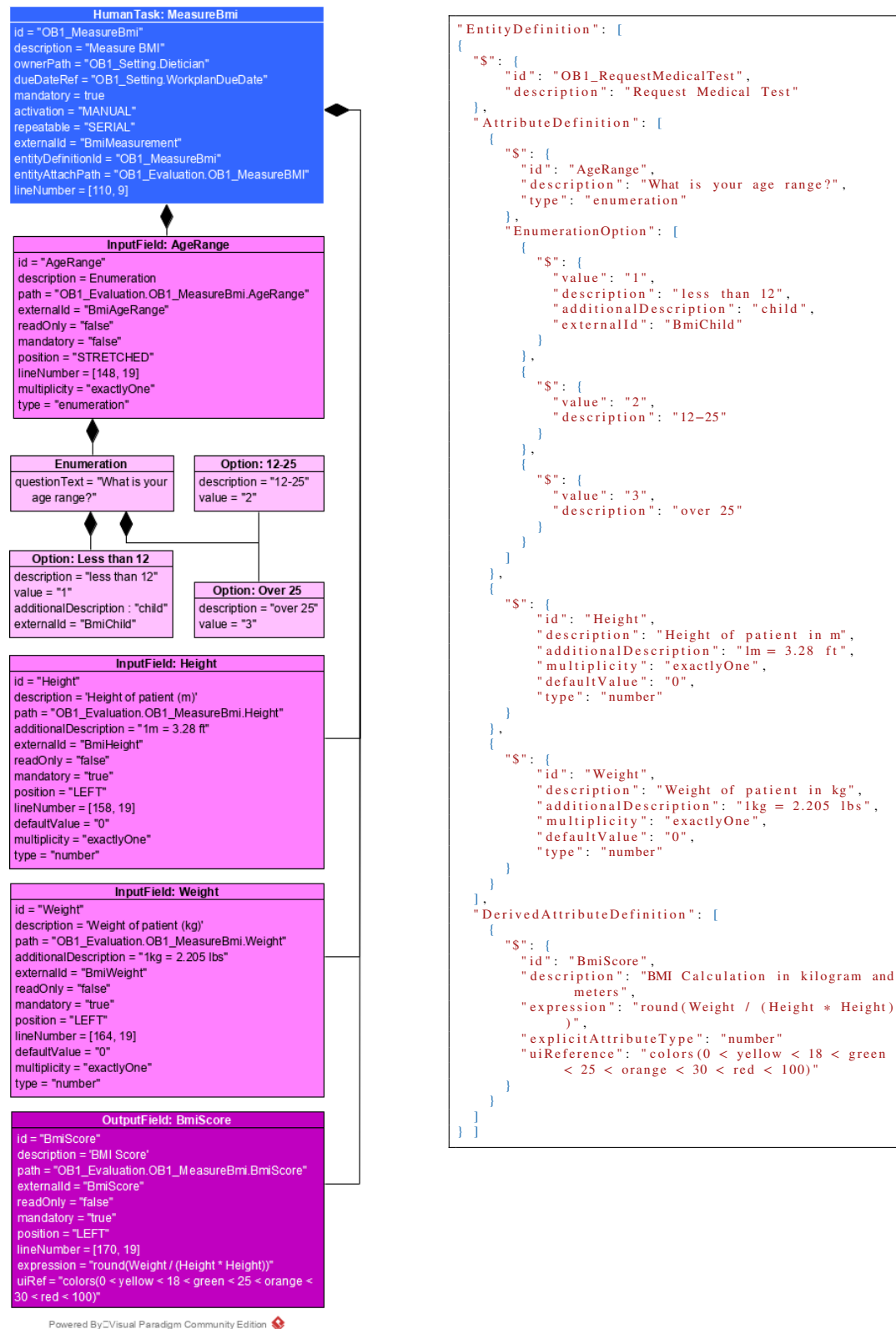


Figure 6.14: Example of compiling a interpreted HumanTask (left) into SACM JSON structure of EntityDefinition (right). The Attribute-related properties of Input/OutputField in the Stage form their AttributeDefinitions.

6.7 Model Visualization

A successfully compiled CP returns its SACM JSON structure to the Acadela frontend, which analyzes the CP elements and applies the corresponding template to render them as *nodes* in a graph. Additionally, the Acadela visualizer examines Preconditions to represent transition conditions as *links* between two *nodes*. This section first describes the *node* templates defining the graphical notation of each CP element, followed by creating nodes and *links* from the CP JSON structure. Figure 6.15 illustrates an example of a CP visualization.

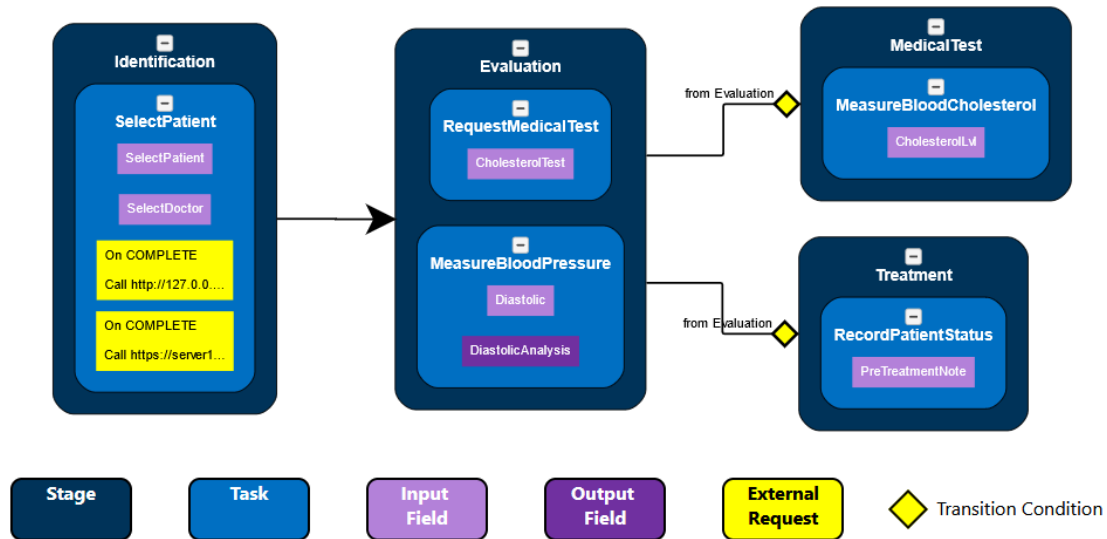


Figure 6.15: Example of a CP Visualization in Acadela using the GoJS tool.

6.7.1 Graphical Notation Definition

Defining Templates of Visual Elements: Acadela leverages the GoJS Group, Node, Shape, and Panel definitions to design the appearance of CP elements. Specifically, the Group template configures an object containing child elements, such as Stages and Tasks. A Node template shapes the layout of InputFields and OutputFields, as they are the leaf elements in a CP. The Shape specifies the background color and form (e.g., rectangle, circle) of boxes representing Groups or Nodes. The Group template uses a Panel to define the internal design, such as the arrangements of child elements, font style of labels as a TextBlock, expandability of the Group (i.e., minimize and expand), and padding. Acadela defines a Group template with a *RoundedRectangle* Shape and Panel to render the name of a Stage or Task. Meanwhile, a Node template displays HttpHook and Input/OutputField as a *Rectangle* Shape.

GoJS Data Binding: A remarkable feature of GoJS is binding its object properties with the attributes of an element in the domain concept. For example, to render the label of a Stage, Task, or Input/OutputFields, GoJS can bind the *text* property of a TextBlock with the *label* attribute of CP elements.

Additionally, GoJS can create a new property to map with an attribute of CP elements. For example, Acadela can insert the *line number* of a CP object into the GoJS Groups or Nodes, so the IDE knows which *line* shall be focused when double-clicking a Group or Node.

Defining Link Templates: Acadela renders Preconditions in Stage and Tasks as a GoJS Link between two graphical elements. If a Precondition does not specify any transition condition, Acadela sets the *link* as a black line. Otherwise, the *link's* endpoint has a yellow diamond shape at the activated Node, with the ID of the prerequisite Stage or Task written above the endpoint. Acadela manipulates the style of the endpoint by specifying the *toArrow* and *fill* properties, which respectively define the shape (e.g., Standard or Diamond) and background color.

6 Implementation

GoJS Graph Components: A GoJS graph comprises two lists of *nodes* and *links*. The *nodes* and *links* are JSON objects containing the rendering properties defined in the corresponding template.

Considering the nodes list, GoJS determines how to render a node object as a Group or Node from the boolean *isGroup* attribute. Furthermore, Groups and Nodes must have a unique ID, with sub-Groups (Tasks) and Nodes (Fields) having an attribute *group* that stores the parent Group ID. This group attribute instructs GoJS to render sub-Groups or Nodes in the parent Group element.

6.7.2 Rendering CP Elements

Stage and Task Visualization: When the Acadela frontend receives the valid SACM CP meta-model, it analyzes the JSON structure to retrieve all Stages and Tasks. Next, Acadela extracts the Stage and Task ID and *line number* to create a Node with a unique key. A Task Group has an additional *group* attribute storing the ID of the parent Stage. Finally, Acadela creates *bgColor* and *textColor* attributes to define the background and text colors. Listing 6.10 and 6.11 demonstrate the GoJS Group object structure of Stage and Task, respectively.

```
1 const stageNode = {
2   key: stage.$.id,
3   label: stage.$.id,
4   // background color is dark blue
5   bgColor: GRAPH_COLOR_CODE.STAGE,
6   textColor: "white",
7   isGroup: true,
8   lineNumber: stage.$.lineNumber['0']
9 };
```

Listing 6.10: Structure of a Stage Group object constructed from the SACM CP meta-model.

```
1 const taskNode = {
2   key: stageId + "_" + task.$.id,
3   label: task.$.id,
4   // background color is light blue
5   bgColor: GRAPH_COLOR_CODE.TASK,
6   textColor: "white",
7   group: stageId,
8   isGroup: true,
9   lineNumber: task.$.lineNumber['0']
10};
```

Listing 6.11: Structure of a Task Group object constructed from the SACM CP meta-model.

GoJS maps the Groups objects to the Group template in Listing 6.12 for rendering Stages and Tasks. First, line 2 instructs GoJS to create a *lineNumber* Node property bound to the *lineNumber* attribute of a Stage or Task. Lines 4 to 11 define the styling, with *fromSpot* specifying that outbound links start from the right side of the Group. Meanwhile, the *toSpot* property states that inbound links end at the left side of the Group. Lines 12 to 15 define the Group shape as a rounded rectangle. Line 13 is a data binding from the *fill* property to the *bgColor* attribute that states the background color. Line 14 states the curve degree of the four corners.

Next, lines 16 to 31 define how to render the Group content. Line 18 enforces the Group to be collapsible and expandable. Lines 19 to 27 express the style of the Group label (lines 20-24), with foreground color and message at lines 25 and 26. Finally, line 29 specifies the padding on the four sides of the Group.

```
1 static constructGroupTemplate = ($) => {
2   return $(go.Group, "Auto",
3     new go.Binding("lineNumber").makeTwoWay(),
4     {
5       alignment: go.Spot.Center,
6       layout: $(go.LayeredDigraphLayout,
7         { direction: 180, columnSpacing: 10 }
8       ),
9       fromSpot: go.Spot.RightSide,
10      toSpot: go.Spot.LeftSide,
11    },
12    $(go.Shape, "RoundedRectangle",
13      new go.Binding("fill", "bgColor"),
14      { parameter1: 20 },
15    ),
16    $(go.Panel, "Vertical",
17      { defaultAlignment: go.Spot.Top },
18      $("SubGraphExpanderButton"),
```

6 Implementation

```
19     $(go.TextBlock, // Stage or Task label style
20     {
21         font: "Bold 12pt Sans-Serif",
22         verticalAlignment: go.Spot.Top,
23         margin: 2
24     },
25     new go.Binding("stroke", "textColor"),
26     new go.Binding("text", "label")
27     ),
28     $(go.Placeholder, { padding: 10 } )
29 ) // End of Panel
30 ); // End of Group Template Definition
31 };
```

Listing 6.12: The GoJS Group Template to represent Stage and Task elements.

Control Flow Visualization: During the construction of Stage and Task Groups, Acadela creates Link objects by extracting the prerequisite Stage or Task ID and transitioning *condition* in Precondition(s). If the Precondition does not have any conditional expression, the Link object sets the *from* property as the prerequisite Stage or Task ID and the *to* property as the current Stage or Task ID. Additionally, the Link object sets the shape of the destination endpoint as a black arrow by configuring the *toArrow* property as "Standard" and the *fill* property as "black". By default, a Link starts from the right side of a Group and arrives at the left side of the destined Group. However, for repeatable Stages and Tasks, the Link starts at the top left and ends at the top of the same Group. In this case, the Link object sets the *fromSpot* property to "TopLeft" and *toSpot* to "Top".

If the Precondition has a transition *condition*, Acadela creates a *condText* property to display the prerequisite Stage or Task ID on top of the endpoint. Furthermore, the Link object sets the *toArrow* property as "Diamond" and the *fill* property as "yellow". Listing 6.13 and Listing 6.14 demonstrate the structure of a Link object for Precondition with and without the *condition* attribute, respectively.

```
1 linkNode = {
2     // fromProcess: unprefixed ID of
3     // previous Stage or Task
4     from: fromProcess,
5     to: stage.$id, // current Stage ID
6     toArrow: "Diamond",
7     fill: "yellow",
8     condText: "from " + rootElement
9     lineNumber: precondition.$
10    lineNumber[0]
11 };
```

Listing 6.13: Structure of a Link object constructed from a Precondition with transitioning condition.

```
1 linkNode = {
2     // fromProcess: unprefixed ID of
3     // previous Stage or Task
4     from: fromProcess,
5     to: stage.$id, // current Stage ID
6     toArrow: "Standard",
7     fill: "black"
8     lineNumber: precondition.$
9     lineNumber[0]
10 };
```

Listing 6.14: Structure of a Link object constructed from a Precondition without transitioning condition.

Finally, GoJS requires a Link template to visualize the flow controls in the workflow, as illustrated in Listing 6.15. Lines 3-5 define the bound attributes to the Link object. Lines 6 to 12 set the Link rendering behaviors as follows:

1. **Routing:** a Link shall take a detour around a blocking Node. In other words, the Link shall not cross through a Node.
2. **Curve:** the Link has a bridge shape at the intersection with another Link. This configuration helps users to trace the link path easier.
3. **End Segments' Length:** the minimum length of the starting and end segments of the Link.
4. **Corner:** specifies how many times a Link can change its direction while reaching the destination, i.e., the maximum amount of times that GoJS can bend a link.

6 Implementation

Lines 13 to 18 configure the visual appearance of the Links, i.e., the line size and shape of the destination endpoint. Finally, lines 19 to 26 state how to render the ID of the prerequisite Stage or Task above the destination endpoint.

```
1 static constructLinkTemplate = ($) => {
2   return $(go.Link,
3     new go.Binding("fromSpot", "fromSpot", go.Spot.parse),
4     new go.Binding("toSpot", "toSpot", go.Spot.parse),
5     new go.Binding("lineNumber").makeTwoWay(),
6     {
7       routing: go.Link.AvoidsNodes,
8       curve: go.Link.JumpOver,
9       toEndSegmentLength: 30,
10      fromEndSegmentLength: 30,
11      corner: 2
12    },
13    $(go.Shape, {strokeWidth: 2}),
14    $(go.Shape,
15      new go.Binding("toArrow").makeTwoWay(),
16      new go.Binding("fill").makeTwoWay(),
17      {scale: 3}
18    ),
19    $(go.TextBlock,
20      new go.Binding("text", "condText"),
21      {
22        segmentIndex: -1,
23        segmentOffset: new go.Point(-70, NaN),
24        segmentOrientation: go.Link.OrientUpright
25      }
26    )
27  ); // End Link Template
28 };
```

Listing 6.15: The GoJS Link Template to represent Preconditions as control flows of the CP.

Input/OutputField and HttpHook Visualization: Since the three elements do not contain any child object, Acadela defines a separate Node template. The Acadela frontend reads the *fieldtype* attribute in each TaskParamDefinition to determine whether it refers to an InputField or OutputField, thus choosing the appropriate background color. The background color is *light purple* for InputFields, *dark purple* for OutputField, and yellow for HttpHook.

The structure of an Input/OutputField or HttpHook object is similar to the Task object. However, the ID of the Input/OutputField Node object follows the `<StageID>.<TaskID>.<Field.path>`. Using a Field *path* alone as a unique ID is not sufficient because two Fields in different Tasks can refer to the same Setting Attribute or Input/OutputField. Likewise, The ID of the HttpHook Node follows the `<StageID>.<TaskID>.<Hook.url>.<Hook.method>` pattern as the same Task or Stage can call the same API URL but with different HTTP methods. Additionally, the Node label of an Input/OutputField Node is the Field ID, while a HttpHook label contains the triggered *lifecycle state* and the first 15 characters of the URL. Listing 6.16 shows the JSON structure of an Input/OutputField Node, while Listing 6.17 demonstrates the JSON schema of an HttpHook Node.

6 Implementation

```
1 const fieldNode = {
2   key: `${stageId}_${task$.id}_${
3     field$.path}`,
4   text: field$.acadenaId,
5   // light purple background for
6   // InputField
7   // dark purple background for
8   // OutputField
9   color: colorCode,
10  stroke: "white",
11  group: stageId + "-" + task$.id,
12  lineNumber: field$.lineNumber['0']
13 };
```

Listing 6.16: Structure of an Input/OutputField object constructed from a TaskParamDefinition in the SACM CP meta-model.

```
1 const hookNode = {
2   key: `${stageId}_${task$.id}
3     _${hook$.url}_${hook$.
4       method}`,
5   text: `On ${hook$.on}\n
6     Call ${shortenedUrl}`,
7   color: GRAPH_COLOR_CODE.
8     EXTERNALCOMM, // yellow
9   group: stageId + "-" + task$.id,
10  lineNumber: hook$.lineNumber['0']
11 };
```

Listing 6.17: Structure of an HttpHook object constructed from its HttpHookDefinition in the SACM CP meta-model.

GoJS constructs the graphical notations of the Input/OutputFields and HttpHook elements based on a Node template described in Listing 6.18. Line 3 binds a *lineNumber* to the GoJS Node object. Line 4 to 6 defines the appearance of the Node box, which has 1) a Rectangle shape without border, i.e., *strokeWidth* is 0, and 2) the background color is determined by the *color* property of the Node object. Finally, lines 7 to 10 define the Node label appearance with a *margin* of 8 pixels, plus the label message and color defined by the *text* and *stroke* properties, respectively.

```
1 static constructNodeTemplate = ($) => {
2   return $(go.Node, 'Auto',
3     new go.Binding("lineNumber").makeTwoWay(),
4     $(go.Shape, 'Rectangle',
5       { strokeWidth: 0 },
6       new go.Binding('fill', 'color')),
7     $(go.TextBlock,
8       { margin: 8 }, // some room around the text
9       new go.Binding('text').makeTwoWay(),
10      new go.Binding('stroke').makeTwoWay(),
11     )
12   ); // End Shape definition
13 }; // End Node template definition
14 }
```

Listing 6.18: The GoJS Node Template to represent Input/OutputField and HttpHook CP elements.

6.7.3 Double-clicking to Focus on the Code Definition

The Acadela frontend leverages the `addDiagramListener` function of GoJS and JavaScript `CustomEvent` to provide this feature. Specifically, the GoJS Graph calls the `addDiagramListener` to register a `CustomEvent` named `graphClick` containing the line number of the GoJS Node. Afterward, the GoJS Graph dispatches this event, which reaches the Acadela IDE that registers a `graphClick` event Listener. Upon receiving the event, the Acadela IDE calls the Monaco `revealLineInCenter` to focus the IDE on the line number of the double-clicked Node.

7 Evaluation

This chapter presents the design, execution, and discussion of the DSL expressiveness and usability evaluations. First, Section 7.1 describes our evaluation approach. Next, we discuss the setup and results of the two assessments in subsequent Sections 7.2 and 7.3. Finally, Section 7.2.5 and 7.3.7 present the analysis of the result, while Section 7.4 reflects the DSL's limitations drawn from the two evaluations.

7.1 Evaluation Approach

The process of our user study resembles the approach of Faber (2019), which adapts the guide of case study research from Robert K. Yin (2009, p. 107). Figure 7.1 illustrates the activities conducted in our user study.

7.1.1 Define Evaluation Goals and Scopes

The primary goal of the user study is to examine the applicability of Acadela in modeling a variety of CPs while possessing characteristics of a user-friendly modeling tool for modelers. In other words, the evaluation aims to find the answers for RQ3 and RQ4 regarding *expressiveness* and *usability*.

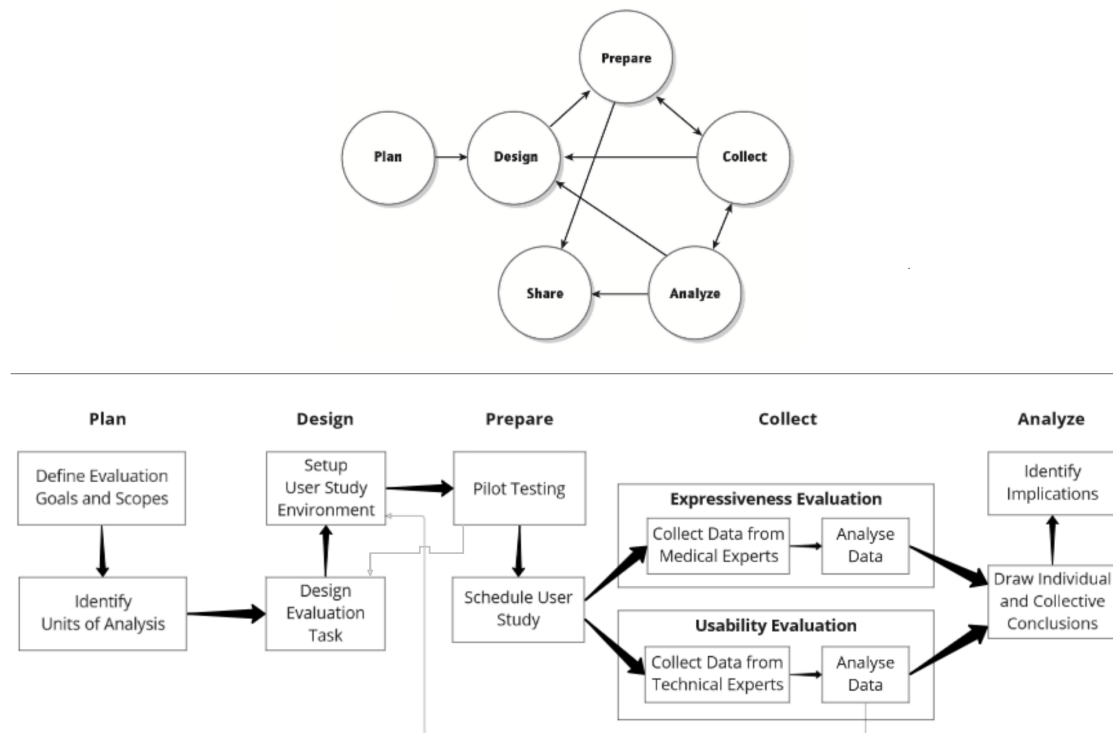


Figure 7.1: Suggestions to conduct user study in research from Robert K. Yin (top) and activities in designing Acadela user study (bottom).

Research Question 3

Can the DSL model Clinical Pathways from **different medical fields** with **diverse complexity** while being **understandable** to **clinical experts**?

We addressed **RQ3** by modeling CPs used in the daily routines of medical experts with the DSL. If Acadela can construct CPs with *different degrees of complexity* and *medical fields* accurately, this implies *the DSL has the potential to model diverse medical processes*. As a result, Acadela can support medical professionals and modelers to develop, execute, and analyze their treatment procedures.

Following this direction, our study seeks to model CPs in any medical field without restriction to any particular treatment or department.

Research Question 4

Do **modelers** regard the *DSL* and the *development environment* **user-friendly** and **learnable** when modeling CPs?

Since the primary users of Acadela are *modelers*, the **usability** and **learnability** of the language significantly influence their *productivity* and *user experience*. For this reason, the DSL evaluation invited technical staff working in medical institutions to model and debug CPs. By experiencing the features of Acadela, the experts can give empirical feedback on the user-friendliness and effectiveness of the CP modeling and error handling capability.

7.1.2 Identify Units of Analysis

Expressiveness Evaluation: The evaluation target is CPs used by medical experts in different medical facilities. The selected CPs shall contain *complex control flows* to test Acadela's ability to model *unpredictable* and *adaptive transition conditions* based on the patient's medical status. Furthermore, the chosen CPs shall contain *repetitive activities* to test the correct functioning of workflow and control flow in different task's iterations.

Usability Evaluation: Due to the limited time budget of technical staff, our experiment examines the construction of a simplified CP with all modelable elements. This setup enables modelers to experience all the features of Acadela and have an overview of the language's capability. As a result, we created a hypertension CP containing missing CP elements, thus, the task of each participant is to define the missing CP elements according to a requirement description. The result from this modeling task implies the practicality and user-friendliness of Acadela in constructing CPs.

Furthermore, our evaluation also considers the understandability of error messages (EMs) in helping modelers correct their mistakes. For this reason, we developed a scenario that includes all catchable syntax and semantic error types. Solving all the bugs implies that Acadela EMs are user-friendly to assist modelers in handling semantic and syntax errors efficiently.

7.1.3 Design Evaluation Tasks

Expressiveness Assessment Tasks Before the interview with each medical expert, we executed and visualized their CP model in the SACM GUI. Then, when the evaluation began, a new case was created from the CP model and presented to the expert. We manipulated inputs from the first stage to show how the workflow triggered different phases and activities. The medical experts then examined the accuracy of the CP and gave us feedback regarding the strengths, limitations, and applicability of Acadela.

Usability Assessment Tasks The evaluation consists of three phases. The experts first attended a *training session*, in which we 1) introduced the motivation of our research and the concepts of modelable

7 Evaluation

elements in SACM and 2) explained how to model CP elements using the Acadela IDE with a modeling tutorial. Next, in the *modeling session*, the experts evaluated the usability of Acadela by defining Stages, Tasks, *control flow*, and *external request communications* of an incomplete hypertension CP.

After finishing the above tasks, the participants were asked to validate their code and fix any bugs indicated by the EMs. Eventually, when their code was error-free, we asked the experts to submit their CP model and begin a *debugging session*.

The final tasks of the evaluation are validating and fixing a CP model containing six syntax and five semantic errors from their EM.

7.1.4 User Study Setup

For both expressiveness and usability evaluation, we created questionnaires with Google Forms to store Likert-scale statements for quantitative evaluation and free-text questions for qualitative assessment. We collected the data by asking the participants to fill out this form after conducting their evaluation tasks.

Deployment for Expressiveness Evaluation Our study deploys Acadela and SACM ecosystem on a PC because healthcare facilities and research institutions may have strict firewall policies that prevent self-signed HTTPS or unsigned HTTP webpages. For this reason, we install the Acadela IDE and backend compiler, SACM web application, SACM engine, Sociocortex, and its database on a laptop. Finally, during the interview, we show how the CP workflow operates from our laptop.

Deployment for Usability Evaluation Similar to the setup of the previous evaluation, we deployed the Acadela-SACM ecosystem on a laptop. However, we use *ngrok*¹ to tunnel the Acadela frontend IDE and backend compiler. Consequently, each component is publicly accessible with its *ngrok* tunneled URL. Henceforth, the technical experts can load the Acadela IDE on their browser. Because *ngrok* produces a new URL in every tunneling setup, we manually configure the tunneled URL of the Acadela backend compiler in the Acadela IDE's setting. With this setup, the expert's browser can send CP validation or compilation requests from their Acadela IDE to the backend compiler of our laptop.

During the evaluation, the firewall of several participants' institutions blocked access to our Acadela IDE webpage. In these cases, we enable the Remote Control feature of Zoom. Hence the participant could directly use the Acadela IDE on our laptop. However, one disadvantage of this setup is a slight latency when typing or dragging the mouse in the Acadela IDE.

7.1.5 Pilot Testing

To prepare for the **usability evaluation**, we tested the assessment tasks and user study setup with three research assistants in Computer Science, who have background knowledge in modeling and programming. These pilot testing sessions gave us insights into the intensity of the tasks. As a result, we optimized the evaluation further by removing duplicated activities, such that *participants model each type of CP element once* throughout their assessment. The pilot testers also shared with us the concepts that need further explanation and their perception regarding the usability of Acadela. This information helps us improve our training materials, evaluation tasks, and estimate possible concerns of our participants.

7.1.6 Schedule User Study

To recruit the participants, our study invited the medical and technical experts who cooperated with us in previous projects or within our network circle. Furthermore, we also invite medical professionals working in clinics or hospitals. First, we expressed our research motivation and briefly described the assessment tasks. Then, if an expert agreed to join the evaluation, we proposed several time slots and decided the

¹<https://ngrok.com/>

meeting time accordingly. Finally, we sent a Zoom meeting invitation to the participant if they prefer online meeting. Otherwise, the interview session will occur at their workplace.

7.1.7 Data Collection

After the experts finished their assessment for each evaluation session, we collected the qualitative and quantitative data from 1) Interviews, 2) Observations of system interactions, and 3) the Google Form.

In the expressiveness evaluation, we gathered data concerning the accuracy of the CP's workflow and execution. For the usability evaluation, we collected data regarding the *ease of use*, *learnability*, *effectiveness*, and *applicability* of Acadela and its error handler. Specifically, we asked participants whether:

1. Defining and editing CP elements using the Acadela syntax is straightforward
2. EMs help participants identify and fix semantic and syntactic errors
3. EMs are readable, accurate, consistent, and supportive.

Furthermore, to obtain an overview of Acadela's usability, we included the *System Usability Scale (SUS)* questionnaire in the evaluation. The SUS score reflects the overall usability of Acadela.

7.1.8 Draw Individual and Collective Results

After each evaluation, we identify trends and indications from the results and observations of the participant's performance or feedback. At the end of all evaluations, we collectively analyzed the data to assess the expressiveness or usability of Acadela. Furthermore, we also derive the implications from the results.

In the expressiveness evaluation, we focus on whether medical experts consider Acadela to construct workflow elements such as *Stages*, *Tasks*, and *stage transitions* correctly. Additionally, we asked whether the system correctly visualized the medical data of patient. Finally, to assess the applicability of CP models, we asked medical experts whether the CP execution accurately provided the necessary treatment. Our study also collects the experts' feedback on how to improve the CP to provide practical support for their treatment.

On the other hand, the usability evaluation focuses on whether Acadela is user-friendly and learnable to technical staff, particularly those with basic programming knowledge. Furthermore, we would like to explore the applicability of Acadela from the experts' perspective. Hence our study collected suggestions on additional CP elements that Acadela should model and whether the DSL can construct CPs used in their medical institutions.

7.1.9 Identify Implications

From the overarching analysis of both expressiveness and usability evaluations, we discuss the limitations and potential of Acadela for each aspect which are summarized in the Limitation and Discussion chapters.

7.2 Expressiveness

We modeled five CPs used in the daily routines of six medical professionals. Afterward, we interviewed the professionals to evaluate the accuracy of the modeled CPs. Our focus is on the correctness of *Stages*, *Tasks*, and *Stage Transitions*. The evaluation goal is to examine the potential of Acadela in accurately constructing the workflow of *various treatment procedures* of *different complexity*. To identify potential improvements, the study also collects feedback about the strengths and shortcomings of Acadela from the perspective of clinical experts.

7.2.1 Population

To facilitate the practicality of the research, our study only considered medical professionals who conduct or research the treatment of diseases at medical or research institutions. We contacted 16 medical professionals and received the voluntary participation from one surgeon, gynecologist, physiotherapist, TCM practitioner, and two psychotherapists. Hence the response rate is moderate at 37.5 percent. All of the participants worked in Germany at the time of the study. The age of the participants ranges from 20 to 60 years old. Table 7.1 lists the background information of the participants.

ID	Department	Years of Experience	Workplace Environment
ME1	Gynecologist	More than 30 years	Clinic
ME2	Traditional Chinese Medicine	More than 25 years	Clinic
ME3	Physiotherapist	4 years	Clinic
ME4	Dermatologist	3 years	Hospital
ME5	Psychotherapist	3 years	Medical Research Institution
ME6	Psychotherapist	3 years	Medical Research Institution

Table 7.1: Background of medical experts participated in the expressiveness evaluation.

7.2.2 Modeled Clinical Pathways

From the contact with six medical professionals, we collected and modeled five CPs with different complexity, ranging from linear to dynamic, adaptive treatment procedures used in their daily routine. Three CPs are linear descriptions of the workflow with repetitive *stages* and a forward variable flow, while the other two guide the experts' decisions with data from the previous steps. The cervical cancer screening CP has *backward variable flows* (i.e., repeatable procedure), and *compound transition conditions*, which are valuable to demonstrate the language ability in modeling complicate *transition conditions* and dynamic workflow for *unpredictable*, *adaptive*, and *personalized* treatment processes. Each CP ends with a discharge of the patient.

COPD Breathing Exercise In Chronic Obstructive Pulmonary Disease (COPD) treatment, physiotherapists instruct patients to conduct breathing exercises to improve their lung condition. Figure 7.2 shows the procedure for conducting the exercises. First, before the breathing exercises, patients specify several *health conditions* such as their stress levels, breathing difficulty, sleeping condition, the presence of mucus and whether the patient can cough the mucus out (See Figure 7.2a). The clinic used a *Rating of Perceived Exertion* (RPE) scale to measure the intensity of these medical conditions (CDC, n.d.).

Afterward, patients start to conduct a breathing exercise. During the session, physiotherapists check whether the patient *inhales and exhales properly*, *sits upright*, and *does not raise their shoulder excessively* (above 1 cm). After finishing the exercise, the physiotherapists can ask the patient about their medical state again for further evaluation (See Figure 7.2b (Cavusoglu, 2021)). The exercise sessions can be repeated several times every week. Figure A.1 of Appendix A.2 visualizes the CP workflow defined in Acadela.

Selection of Antipsychotics for Schizophrenia The care goal of this CP is to help psychotherapists and patients cooperate to decide which medications are suitable to avoid or mitigate unwanted side effects chosen by the patients (Siafis et al., 2022). First, patients read the textual description and video about schizophrenia, aspects of treatment using antipsychotics, and the *shared decision-making* (SDM) practice. Healthcare practitioners and former users produced these materials. Next, psychotherapists collect the patient's *preferences* (e.g., unwanted side effects, previous experience with antipsychotics), *medical condition*, and *personal data* (e.g., age, sex, duration of illness) to the patient profile.

7 Evaluation

a)

Please answer the following questions

1. Do you feel stressed or short of breath today? (Green: Not at all - Red: So much)
2. Do you struggle to breathe? (Green: Not at all - Red: So much)
3. How was your sleep last night?
 - Good
 - Medium
 - Bad
4. Do you have mucus today?
 - Yes
 - No
5. Can you cough your mucus out?
 - Yes
 - No

b)

1. Do you still feel stressed or short of breath? (Green: Not at all - Red: So much)
2. Do you still struggle to breathe? (Green: Not at all - Red: So much)
3. Can you cough the mucus out better after the exercise?
 - Yes
 - No

Next Finish

Figure 7.2: Execution procedure of the breathing exercise in a web application. a) The patient answers questions about health status prior to the exercise. b) A post-questionnaire to record the patient’s condition after the exercise (Cavusoglu, 2021).

Henceforth, the psychotherapists and patient organize a *therapy session* to discuss the *efficacy* and *side effects* of *antipsychotics* on the patients. In this session, they jointly analyze a forest plot produced by a SDM-assistant web application that shows the *efficacy* and *risks* of chosen side effects (See Figure 7.3). The application expresses the effects of each antipsychotic using a line with a median value. The start and end of the line denote the lowest and highest relative risk, respectively.

In the third step, the patient and physiotherapist select or exclude antipsychotics based on the goal and preferences of the patient. Further discussion between the patient and medical experts about their selections is possible. Finally, they conclude the selected medications and save the session. The *therapy sessions* can be repeated *multiple times* in the CP. Figure A.2 of Appendix A.2 visualizes the CP workflow defined in the Acadela language.



Figure 7.3: Forest plots present the effect sizes on efficacy, fatigue, and weight gain for comparisons of antipsychotics and placebo. Relative risks are the measurement unit for effect size, except for weight gain which uses mean differences in kilogram (Siafis et al., 2022).

Diagnosis of Class II Smoke Inhalation The CP submitted by the surgeon concerns the diagnosis of Class Two Smoke Inhalation (Karpov, 2018). Figure 7.4 presents the treatment activities of the CP. First, dermatologists examine immediate signs, potential signs, symptoms, and the results of a laryngoscopy. Depending on the outcome, the medical expert can suggest further treatment for the patient. The CP consists of seven stages. After admitting a patient, the situation of the patient is evaluated step by step and conditionally decided whether the patient is diagnosed with class II smoke inhalation. Figure A.3 of Appendix A.2 visualizes the CP workflow defined in the Acadela language.

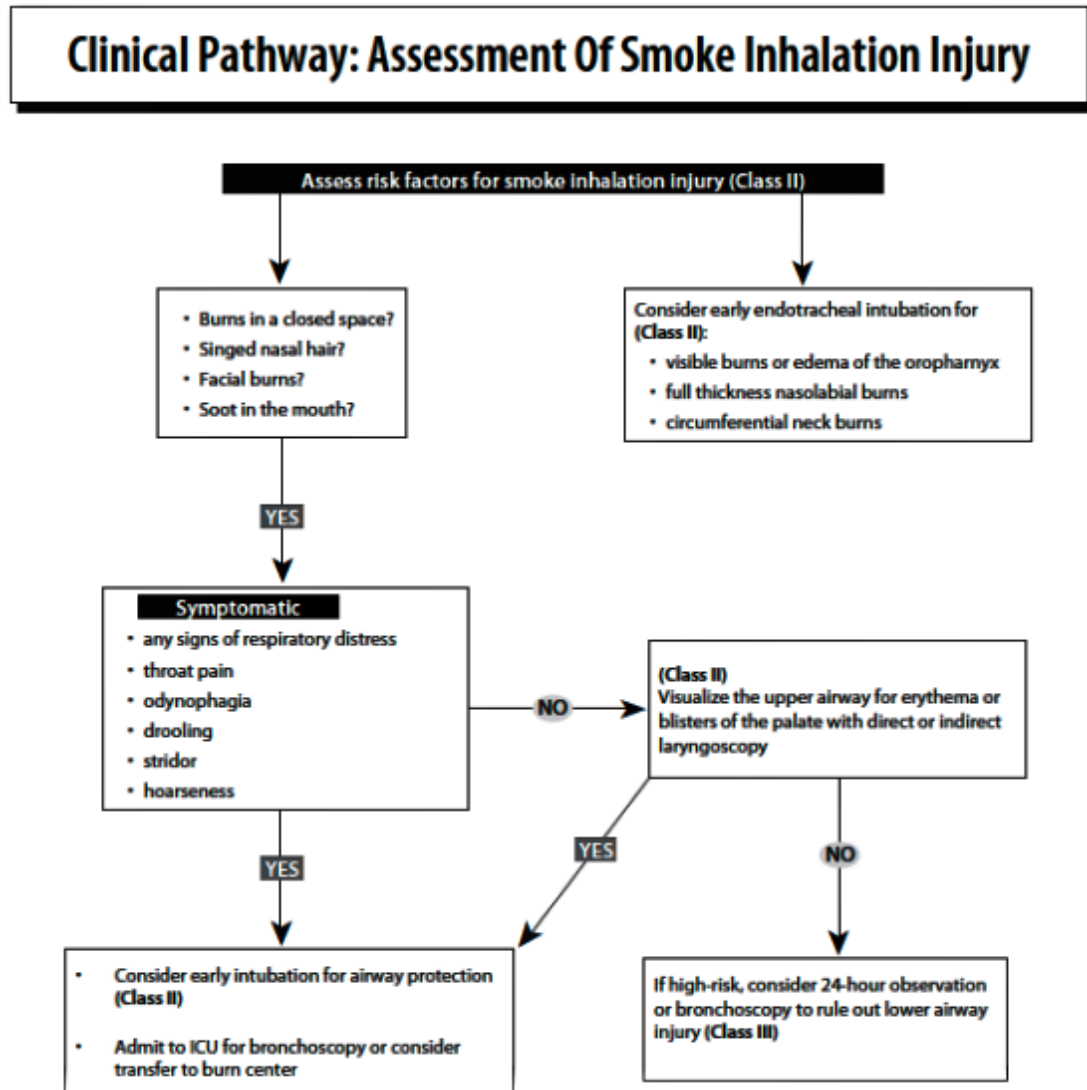
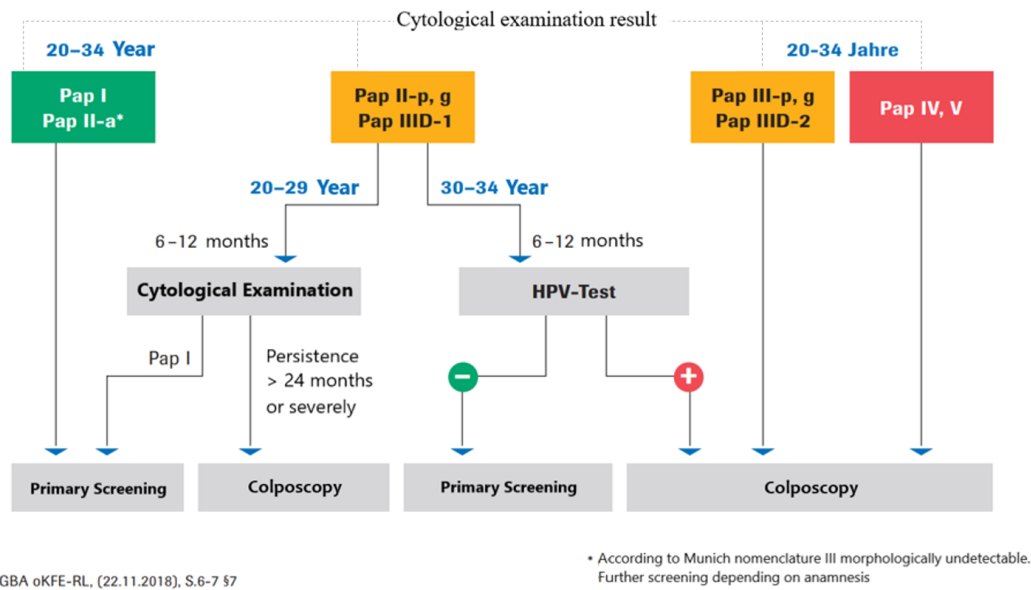


Figure 7.4: CP of Smoke Inhalation Injury Assessment. The process is constraint-driven based on the symptoms (Karpov, 2018).

Cervical Cancer Diagnosis Figure 7.5 describes a CP to diagnose cervical cancer from the gynecologist (Roche Diagnostics, 2018). The CP takes the 1) patient's age and 2) the result of cytological and/or HPV tests to determine the necessity and time of the next colposcopy, or advise the patients to follow a routine screening procedure (e.g. every three, six or twelve months). Colposcopy is a cervix test that uses a magnifying instrument (colposcope) to inspect abnormal cervical cells and take a sample (biopsy) from them (Roche, 2016). The screening procedure involves sampling cervical cells and analyzing them using cytology (Roche, 2016), which is an exam to diagnose or screen cancer in single cells (John Hopkins Medicine, n.d.). Figure A.4 of Appendix A.2 visualizes the CP workflow defined in Acadela.

Cervical cancer screening guideline

Evaluation algorithm in annual cytological screening for women aged 20-34 years



Evaluation algorithm in co-testing for women 35 years and older

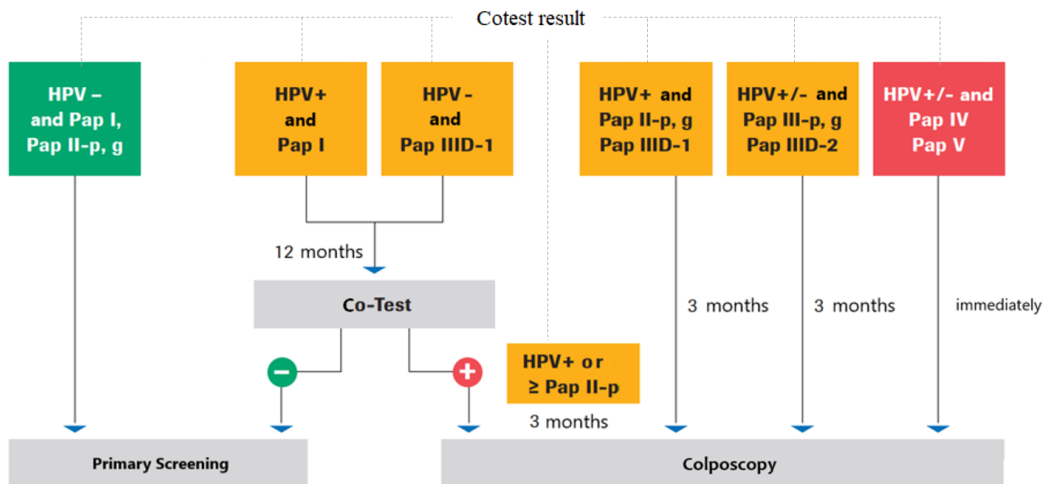


Figure 7.5: CP of Cervical Cancer Screening. The guideline suggests different treatment processes depending on the patient's age (Roche Diagnostics, 2018).

Chronic Headache Treatment Through contacting a TCM practitioner, we studied the treatment of chronic headaches using (heated) *massage*, *acupuncture*, and *Guasha* techniques. First, the practitioner observes the patient's *behaviors* (e.g., handshake, or walking style) and *mental state* since the greeting moment. This initial observation reveals clues about the potential physical or mental causes of the headache. Next, the practitioner asks questions regarding the headache, such as the history and frequency of the pain, emotions or actions that worsen the headache, etc. These question reveal the factors (e.g., cold weather, anxiety) causing the pain. Afterward, the practitioner discusses his treatment approach with the patient. If the patient consents, the practitioner performs the proposed techniques. Otherwise, both sides discard the disagreed method(s).

7 Evaluation

This CP demonstrates the application of a *customizable graphical representation template*. Specifically, we render the massage positions on a human head in a SVG image based on the selected massage areas of the practitioner. This visualization helps the patient see which body locations will be massaged and discuss any potential concern with the doctor before proceeding to the treatment. Figure 7.6 shows the visualization of chosen massage positions in SACM. Listing A.9 in Appendix A.3.2 shows the code to define the *InputFields* of the massage positions and the *OutputFields* to declare the SVG, Javascript style, and the final graphic. Figure A.5 of Appendix A.2 visualizes the CP workflow defined in Acadela.

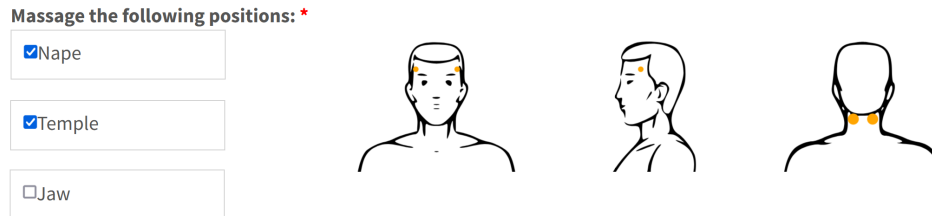


Figure 7.6: Illustration of InputField values (massage positions - left) affect the output image visualization (right)

7.2.3 User Study Setup

Software Deployment We host the Acadela and SACM components in our computing devices and show them offline to avoid being blocked by strict firewall policies in medical institutions. Due to the Corona pandemic, depending on the participant's condition, we presented the CP model, collected the survey, and interviewed them on-site or via Zoom.

Survey Design We use Google Forms to develop an online questionnaire with free text questions and *Likert scale* ratings of statements about the *correctness* of the modeled *workflow items* and *control flow*, as shown in Figure 7.7. The answers from the first three statements reveal the *accuracy* of modeling CP elements, while the last two refer to the appropriate *visualization* of CP elements. We grade each remark on a scale of one (strongly disagree) to five (strongly agree).

7.2.4 Result

After modeling the CPs, we presented the models to the medical professionals using the web application of SACM. First, we created a patient case from the CP model. Next, we activated *Stages*, *Tasks*, and possible *Stage Transitions* by manipulating the medical inputs of the case. We then asked the participants to evaluate how accurately the system executed their CP. Furthermore, we also collected qualitative feedback to learn the expectations of medical professionals.

Quantitative Accuracy Evaluation We presented a questionnaire to the participants with free text questions and *Likert scale* ratings of statements about the *correctness* of the modeled *workflow items* and *control flow*. The answers from the first three statements reveal the *accuracy* of modeling CP elements, while the last two refer to the appropriate *visualization* of CP elements. Figure 7.7 shows the assessment of the medical professionals on the accuracy of the modeled CP.

The quantitative result suggests that the *system constructs CPs elements accurately*. All participants *agreed* or *strongly agreed* with each statement. This implies that Acadela has the potential to reconstruct various medical procedures precisely, such that the SACM system can correctly show and execute the treatment process accordingly.

7 Evaluation

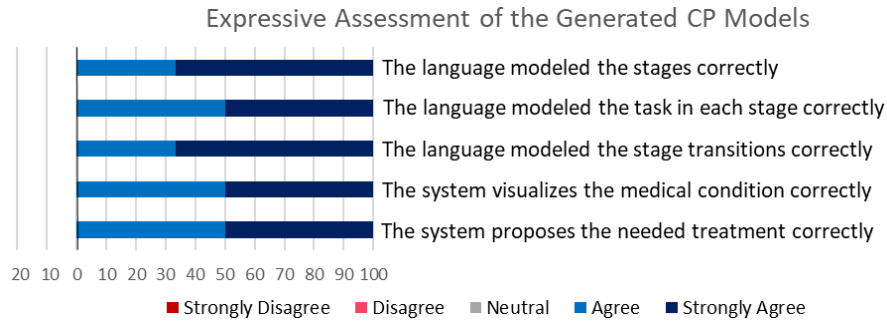


Figure 7.7: Rating of medical professionals on Statements regarding the accuracy of modeled CP elements (stage, task, and transition condition), visualization, and treatment process.

Qualitative Feedback from Medical Experts Figure 7.8 summarizes the feedback we collected during the interview from the participants. We categorize the participants' quotes primarily based on the **expressiveness** and **applicability** of CP models. Overall, the professionals considered the modeled workflow as correct. However, our CP model does not reconstruct specific processes precisely, like displaying the waiting period before the next cytological test (QE5) in the cervical cancer screening CP.

In addition, we learned several treatment variations after modeling the original CP, so our models did not cover them in the final meeting with the medical professionals. Nevertheless, our models can reflect the variations if we questioned these alternatives in the first contact and modeled them accordingly.

Regarding the applicability, participants state that the CP models need to address a range of variations to be practically applicable in treatments. The TCM practitioner regarded the CP model appropriate to serve as a treatment guideline and storage of medical data.

ID	Med. Prof. ID	Quoted Statements	Indicator	Dimension
QE1	ME1	The period between the tests is important, but they were not well reflected. Show text when the next test is due.	Accuracy	Expressiveness
QE2	ME3	yes, we have a questionnaire and can give a comment individual to the exercise, and what was wrong or right in the exercise		
QE3	ME6	The system covered the medical condition of selecting antipsychotics for the treatment of schizophrenia. The visualization of this condition was appropriate, yet as mentioned above and below more details could be added.		
QE4	ME6	The treatment process is in overall accurate, but as stated before, it is mainly focused on the selection of antipsychotic treatment. Some aspects could be improved in order to cover better this topic. In addition, it can take into consideration the more complicated nature of the treatment of schizophrenia (e.g. adding psychotherapy, etc.)		
QE5	ME6	The stages were modelled correctly, e.g. starting with adding the patient in the system, adding baseline information, selecting antipsychotic (more treatment aspects could be added), and then recording the outcome of the session (in order to be used in the future).		
QE6	ME2	Human behavior is difficult to capture. The process shall capture medical actions in depth. Massage (on the shoulder) can be around an area when people curl or push hard on the shoulder.	Missing CP Variations	Expressiveness
QE7	ME3	I think the questionnaire after the exercises have to be more individual and the questions have to be compared to the exercises, there are different aims we want to reach with different exercises.		
QE8	ME5	The discharge phase should allow to put the patient again in		
QE9	ME6	The main framework and tasks per stage seems to be correct. However, more information and details could be added for each task. For example, baseline information about functioning (e.g. occupation, family, education etc.) was missing. In addition, concomitant medication for comorbidities (e.g. insulin for diabetes etc.) could be added as well as the possibility to add more comorbidities that were not covered. In addition, some information on the clinical status of the patient, e.g. symptomatology (e.g. measured by scales such as CGI-Severity).	Usable	Applicability
QE10	ME2	The model is alright for data recording		
QE11	ME5	I would like to use the language by myself. In urgent cases, patients cannot wait, so I need to modify and record the data immediately. However, I prefer a low-code approach.		
QE12	ME5	I am comfortable with editing a typo in the model		
QE13	ME2	The model can serve as a guideline. However, one shall teach and show the medical students or practitioners how to do it	Complexity Barrier	Applicability
QE14	ME6	The treatment for schizophrenia is multi-dimensional and more complicated.		

Figure 7.8: Excerpts of feedback towards the modeled CPs from medical professionals.

7 Evaluation

Considering suggestions for improvements, the psychotherapist wished to see 1) a combination of an infographic and temporal display of medical data from the task, 2) inclusion of extra personal data from the patient (e.g., occupation, family, education), 3) medical information like *comorbidities* (diabetes) and their *concomitant medication* (insulin) and 4) an evaluation of the potential side-effects of the prescribed antipsychotics by sending a request to an external system.

Overall, the psychotherapists are content with the system's ability to document medical data. Noticeably, the two psychotherapists who have experience in R programming language are willing to fix errors in the code. One participant states their motivations arise from 1) the urgency of executing a correct treatment procedure for the patients, and 2) he may forget to modify the CP afterward due to the intense workload in the clinic.

Likewise, the physiotherapist considers the procedure accurate in general. She remarks on the ability to add individual comments to tell if the patient was doing right or wrong. The physiotherapist states that she is willing to make small changes to the code, such as a typo in the name of a `Stage` or `Task`.

When asked about possible improvements, she would like to see an inclusion of additional exercises and medical practices in the CP, as they help her treat individual cases of COPD patients. Furthermore, she suggested personalizing the post-exercise questionnaires to specify the treatment goals of different exercises. This ability enables physiotherapists to provide more precise treatments for individual patients according to their medical conditions and preferences.

For the CP of the cervical cancer diagnosis, the gynecologist considers the workflow accurate, yet he noted two necessary improvements. Firstly, a properly functioning `Stage` repetition that conditionally unlocks a `Stage` outside of the iteration. This suggestion arises due to a SACM bug that prevents the web page to render the latest stage iteration correctly, although the system accurately records the medical inputs of each iteration. The second suggestion is to display proper advice about the next time for medical re-examination, e.g., after three or six to twelve months.

The quantitative and qualitative results suggest that the medical professionals consider their reconstructed CPs *accurate*. In other words, the system *correctly* models *Stages*, *Tasks*, and *Stage Transitions* in the treatment procedures. However, the participants wish to see further enhancements to their CP in SACM.

7.2.5 Discussion

The results imply that Acadela has the potential to *model CPs from various medical fields accurately*. However, we must consider several remarkable improvements to assist medical experts more effectively. Additionally, we see the potential of using the language as a shared artifact for technical and domain experts to communicate and co-create values.

Complex Modeling Potential Acadela offers a *simplified syntax* to model elements of CPs while reducing the complexity of the original modeling concept of SACM. The language's ability to model *workflow activities* (`Stages` and `Tasks`), *control flow*, *responsibilities assignment*, and *external communication* is promising for constructing real-life executable CPs applied in various medical facilities. This capability enables modelers to reconstruct complex medical procedures by *conditional unlocking* `Stages` based on data from `Tasks` to model the decision-tree structure. Meanwhile, modeling linear CPs is usually sufficient by connecting workflow activities with simple *stage transition condition(s)*. Our CP models demonstrate that technical experts can build accurate CPs from *different medical departments* and *complexity* with Acadela. However, we were not informed about variations during the CP collection phase; thus, our models did not include the handling of all alternative execution paths.

Potential Facilitation of Co-creation In the interview, four out of six medical experts expressed willingness to fix minor mistakes in the executable CPs. This phenomenon suggests that medical professionals can reason about the cause of minor errors after viewing the source code. This phenomenon suggests that technical and domain experts may have a *shared understanding* of the artifact expressed by the DSL, enabling them to discuss the workflow logic.

Integration with Existing e-Health Systems Our study demonstrates that Acadela can seamlessly export CP models in a SACM-compatible format. In other words, *Acadela has the potential to interpret the models in a structure that is processable by an e-Health system*. This could foster collaboration among medical facilities as one Acadela CP model is executable in multiple different e-Health systems without modifying their code base.

Applicability Four out of six experts considered Acadela has the potential to model CPs in e-Health applications. One reason is because Acadela can model control flow to trigger specific treatment activities depending on the patient's condition. Furthermore, sending requests to external systems is useful to enable communication between e-health services necessary for the treatment.

7.3 Usability Evaluation

Besides the capability of modeling various CPs, user-friendliness is also a crucial aspect that influences the modelers' satisfaction, productivity, and user experience. Therefore, we designed a *field experiment* where modelers 1) construct a *Hypertension CP model* using Acadela in their working environment and 2) fix syntax and semantic errors in another CP model based on the error messages provided by Acadela. After finishing the two sessions, we asked the participants to complete their usability evaluation survey and begin the interview process, thus we can quantify the user-friendliness opinions and learn about the limitations, improvements and advantageous features of Acadela.

7.3.1 Population

We invited 21 software developers or medical researchers with technical experience at medical or research institutions to participate in our experiment voluntarily. We received responses from eight staff, comprising five medical researchers, two technicians, and one physician with programming or modeling experience. The response rate is moderate at 33.3 percent. All participants worked in medical research institutions across Germany, Spain, or Italy. Figure 7.2 presents their background information.

7.3.2 Experiment Setup

Similar to the setup of the expressiveness evaluation, we host the Acadela and SACM components in our computing devices. However, the experiment is conducted solely via the Zoom platform. Participants joined our Zoom room from their workplace and used their devices to conduct the experiment.

7.3.3 Experiment Design

The evaluation consists of two phases. The participants first attended a *training* session, in which we introduced how to model CP elements in Acadela. In the *modeling* session, the experts constructed elements of an incomplete hypertension CP and fixed bugs of another CP model by examining their error messages (EM). We controlled the *evaluation tasks* and *bugs* such that all participants modeled the same CP elements and debugged the same syntax and semantic errors. Performing the same activities builds a common baseline to gauge the usability and learnability of the language.

To assess our DSL usability primarily, we conducted a pilot study with three researchers in computer science who are experienced in programming and UML modeling. After that, we analyzed and incorporated the feedback in this phase to further refine the language syntax and experiment procedure.

7.3.4 Training

Language Introduction In about 20 minutes, we explain our research motivation and examples of modelable CP elements. Afterward, we instructed the technical staff on how to model CP elements with

ID	Occupation	Age Range	Expertise (Years)	Programming Experience	Experienced Modeling Language	Location
TE1	Research Assistant	36-45	15	C, C++, Java, JavaScript, Python, R, SQL	XML, UML, BPMN , CMMN	Spain
TE2	Researcher	36-45	2	-	PlantSimulation (Siemens)	Spain
TE3	Research Assistant	36-45	1	C#, Python	BPMN	Germany
TE4	Medical Doctor	Over 46	20	Visual Basic, Pascal, SQL	BPMN	Germany
TE5	Technician	26-35	9	Java, Python, SQL	XML, JSON	Spain
TE6	Research Assistant	26-35	3	C, C++, Java, Python, Boogie, SQL	UML, JSON	Germany
TE7	Medical Doctor & Research Assistant	26-35	2	R	-	Italy
TE8	Research Assistant	26-35	4	R	-	Germany

Table 7.2: Background of technical staff participated in the usability evaluation

Acadela grammar and the auto-complete feature of the IDE. We emphasized that the participants can copy-paste or auto-complete CP items to avoid memorizing the syntax.

Exercise To practice the language concepts, we asked the technical staff to construct the introduced CP elements and attributes in an exercise with the Acadela IDE. The participants can ask for clarification of any concept in this 20-minute exercise.

7.3.5 Modeling

After having a 5-minute break, we invited the technical staff to a 50-minute evaluation session. The tasks are 1) complete a simplified hypertension CP by defining its missing elements, and 2) fix 11 semantic or syntax errors inside a faulty model by analyzing their EMs. We read our script to each participant to inform the experiment procedure and ask them to solve arising problems by themselves and refrain from asking us questions during the evaluation period. Figure 7.9 shows the workflow of the hypertension CP.

CP completion Participants modeled the following missing elements in the CP:

1. Create a *Stage* and *Task* with an *Input Field* to discharge a patient. The *Condition* to discharge is the completion of the *Identification Stage*.
2. Create an *Input Field* to record the numeric *Systolic blood pressure* (BP) value and visualize its severity with color. For instance, apply the **green** color if the *Systolic* value is **below 120**. Add an *Overall Assessment* *Output Field* to conditionally show the BP condition. For example, display a **"Normal"** text if the *Systolic* value is below 120 and the *Diastolic* value is under 80.

7 Evaluation

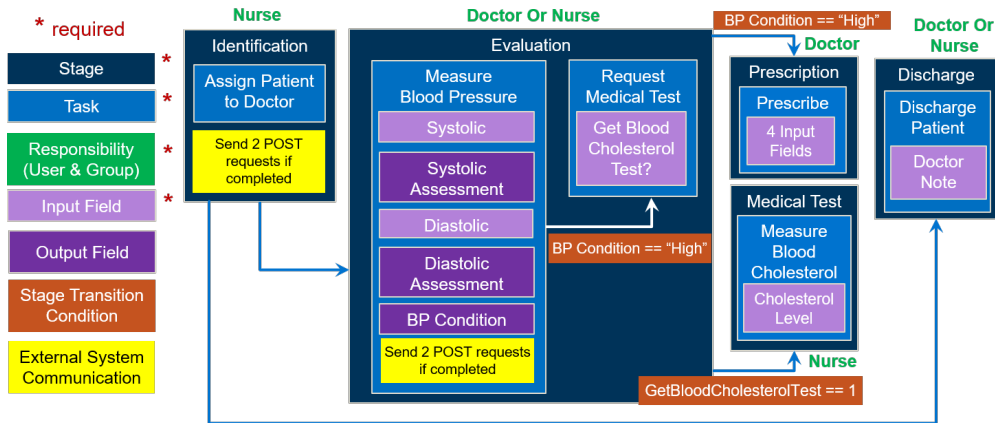


Figure 7.9: The workflow of a simplified Hypertension CP

3. Add a Transition Condition for the *Treatment* Stage such that the patient must have a **"High"** BP condition and already finished the *Evaluation* Stage.
4. Import a *Prescribe* Task from a file path into the *Treatment* Stage
5. Define a POST HTTP *request* to a URL endpoint when the *Measure BP* Task is *completed*. If the *request* fails, display a user-defined EM.

After finishing the above tasks, we asked participants to validate their code and fix any bugs indicated by the EMs. When their code was error-free, we asked the experts to submit the model and begin the debugging task.

CP Debugging In this task, participants validated and fixed a CP model containing syntax and semantic errors. For every bug validation request, the EM includes 1) error cause, 2) line and row number, 3) the code surrounding the bug location, and 4) solution suggestions.

The session finishes when the experts successfully correct six syntax and five semantic errors. Examples of syntax errors are *typos*, *unexpected element or attribute*, *missing parenthesis* in the if statement. We insert semantic mistakes such as *duplicate stage name*, *invalid path reference* to a Task or InputField, and send requests to an *unknown URL* or using *disallowed HTTP methods*.

7.3.6 Result

Quantitative Accuracy Evaluation After the debugging session, we asked the experts to assess the usability of Acadela and its error validation. We presented a questionnaire with statements regarding the 1) Ease of use and learnability, 2) Usability of EMs, 3) Helpfulness of EMs, and 3) The overall usability assessment using the System Usability Scale (SUS). Table A.1 in Appendix A.4 lists the statement in the SUS questionnaire. We used a Likert scale from 1 (Strong Disagree) to 5 (Strongly Agree) to rate the statements. Figure 7.10 summarizes the quantitative evaluation outcome.

The results from chart a) suggest that most technical experts consider Acadela syntax easy to use, understand and learn. Chart b) displays the agreement from participants that EMs accurately explain the source of errors with straightforward language and practical information to help them locate and fix the bugs. In particular, the majority of technical experts strongly agreed that the EMs support them in solving the semantic and syntactic errors as shown in chart c).

The overall SUS score in d) reflects the usability of Acadela. Following the SUS grade ranking of Bangor et al. (2009) from Figure 7.11, with an average score 78.75, Acadela can be classified in the "Good" category. The standard deviation of 15.11 implies that the participants' opinions are highly distinctive.

7 Evaluation

ID	Syntax Usability Statement	ID	EM Usability Statement
S1	The syntax for creating CP elements, i.e. Stage, Task, Form, Field, is straightforward	EM1	The EM were easy-to-understand
S2	Editing CP elements is easy with the language	EM2	I was easily able to locate the source of error using the EMs
S3	Importing external modules is straightforward	EM3	I was able to fix the errors easily using the EMs
S4	The syntax of the language was easy to learn and use	EM4	The language of the EMs is clean and precise
		EM5	The EMs were accurate
		EM6	The EMs were consistent

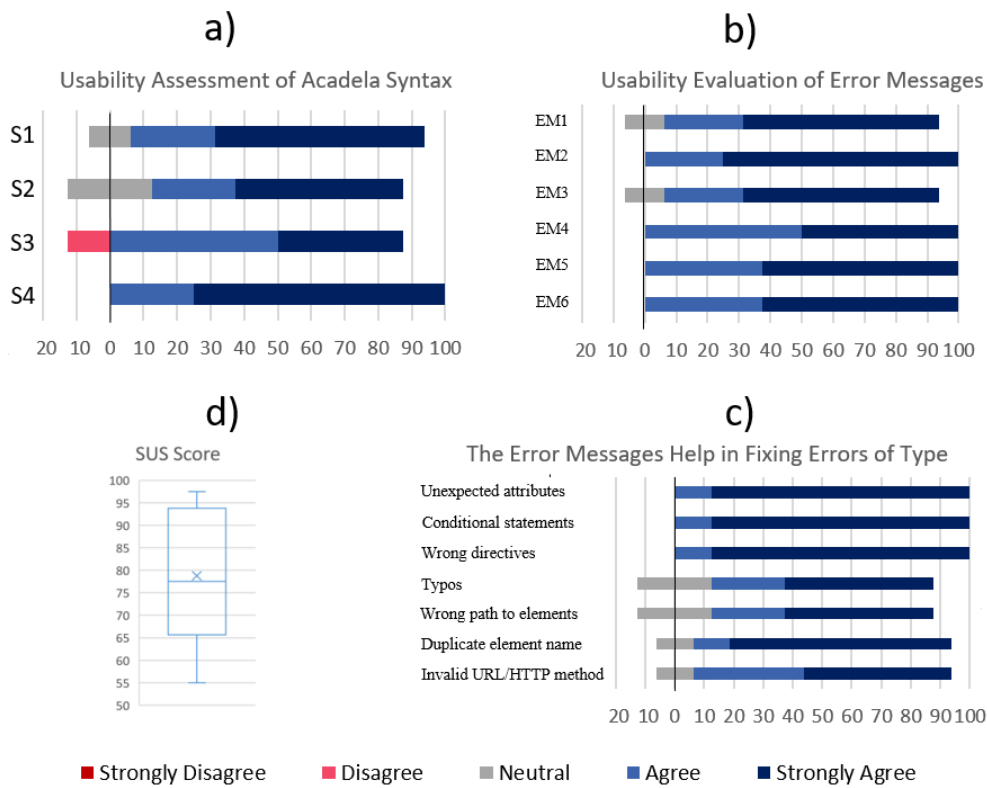


Figure 7.10: Rating of experts on Statements regarding the a) Usability of the Acadela Syntax, b) Usability of EMs, c) Usefulness of EMs, and d) Acadela SUS Score.

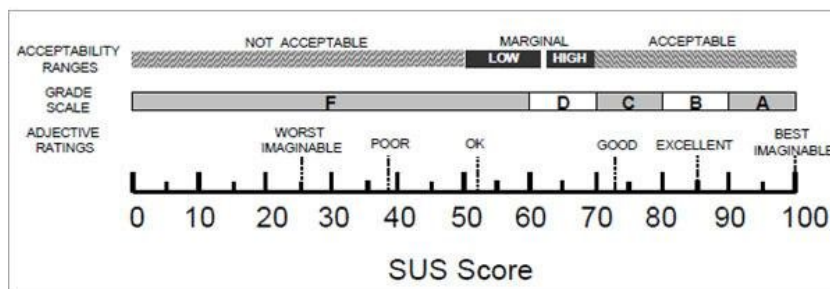


Figure 7.11: Grade Ranking of SUS Score.

Qualitative Feedback We also interviewed the participants to learn their opinions regarding the *applicability* and *user-friendliness* of the language syntax and its EMs. Additionally, we asked the experts for any negative experiences and suggestions for improvements.

Overall, the participants consider the syntax as "simple", "elegant", "easy to learn" for users with competent programming experience. Three technical staff commented that 'the auto-complete feature is supportive'. Three other participants shared an opinion that the language is "fairly easy to use, but this will certainly depend on the background and education of the person in questions". Another participant noted, "Successful repairing errors required basis programming skills/knowledge". The two statements imply that modelers need strong fundamental programming knowledge to apply and debug CP meta-models written in Acadela.

Furthermore, the participants state that EMs are "useful" as they "show the line number and the problem" to help users "pinpoint the error". Additionally, syntax highlighting helps identify potential errors, like "spotting (missing) string quote error".

Regarding limitations, one technical staff suggested "visualizing the model" to provide an overarching view of the CP. Thanks to this suggestion, we developed the CP visualizer for Acadela. The second remark is the auto-complete feature should "show the names of elements in the code". This option is convenient for the users and prevents typing a non-existing CP component.

7.3.7 Discussion

The results imply that Acadela syntax is considerably learnable and usable. Additionally, it also assists modelers in identifying and fixing errors in CP models effectively. However, we need to consider several remarkable improvements to offer further support.

Intuitive Syntax From the usability assessment of the syntax, all participants regard Acadela as easy to learn and use. Most technical staff agree that constructing and editing CP elements is straightforward with the language syntax. When considering the time to finish the modeling and bug fixing sessions, two participants who had basic programming experience spent around 50 and 60 minutes completing their tasks; one participant with modeling experience but limited programming knowledge stopped the evaluation after exceeding the session period; the other three participants with advanced programming experience spent 20 to 30 minutes. The three participants who invested a longer duration for their evaluation considered the language is more straightforward if they could have more time to learn Acadela syntax. This observation implies that Acadela can be user-friendly to modelers with solid fundamental programming skills, while experienced ones can quickly understand the language concepts. The SUS score with the mean of 78.75 and left-skewed data distribution consolidates the good usability result from the majority of the population.

From the qualitative feedback, auto-completion of CP elements has a remarkable impact on the user-friendliness of Acadela. First, it eliminates the need to memorize the syntax or check the Wiki to use the correct definition. Second, an auto-complete saves time because it presents users with options for a minimal or complete construction of a CP item. As a result, the users increase their productivity and experience in modeling CPs.

The mean SUS score of 78.75 with high standard deviation (15.11) and left-skewed data distribution demonstrates the majority of the participants consider the DSL to be user-friendly, while others believe that modelers need strong foundational programming knowledge as a minimum to use Acadela. This result implies that the concise syntax combined with IDE features could increase the productivity of experienced modelers when developing CP models.

Supportive IDE and Error Messages From the qualitative feedback, **auto-completion of CP elements** has a remarkable impact on the user-friendliness of Acadela. First, it eliminates the need to memorize the syntax or consult the Wiki. Second, an auto-complete saves time because it suggests search options to construct templates of CP items quickly.

Helpful Error Handling The population considered the EMs to indicate the source of error accurately, consistently, and comprehensibly. To support participants in fixing various semantic and syntactic errors, Acadela presents the problem statement, line of error, faulty code piece, and suggestions for possible fixes when applicable. The information saves the cognitive effort of participants to reason the cause of errors in the code and make the necessary changes. As a result, the majority of participants agreed that Acadela effectively supports modelers in handling the covered syntactic and semantic bugs.

7.4 Limitations

7.4.1 Need Supportive CP Elements

According to the technical experts, although Acadela can model the crucial and assistive elements necessary for modeling executable CPs, it should also construct CP elements that increase the treatment quality. For example, one expert stated that to support medical professionals in conducting effective treatments, Acadela should include video, report monitoring, and gamification factors in its modeling concepts. Furthermore, Acadela should also provide a summary for each Stage to display critical medical data of the treatment phase.

7.4.2 Support Previewing CP Elements

Acadela does not display the properties and graphical representation of CP elements. For example, when clicking on an `InputField`, the Acadela frontend should intuitively show the attributes of the `InputField` and preview its UI appearance in the SACM system. This feature helps modelers and medical professionals to verify the properties of CP elements and identify improper visual elements before generating the CP meta-models.

7.4.3 Auto-complete CP Elements in Web-based IDE

The Acadela IDE can auto-complete CP elements but not the element's ID. Providing this feature will help modelers type the target element quickly or realize whether it exist in the CP definition's scope.

7.4.4 Dependent on textX Error Handler

Since we rely on the textX syntax and error handler, several edge cases that textX cannot properly catch can result in misleading error messages from Acadela. In these cases, textX does not reveal the exact cause and location of the syntax error. As a result, modelers can be confused in identifying the cause of bugs. However, in the cases we observed, the returned error code snippet location is still near the actual position of the bug. Thus modelers may be able to recognize the source of error.

7.4.5 Mishandling of Special Characters

During the evaluation, one participant who worked in Italy created an unknown character error in the compiler when he typed "fi", which his device converted to a special character. This event suggests that Acadela may not handle unique characters in different languages correctly.

7.4.6 Limited Number of Participants

The numbers of participants in both the expressiveness and usability evaluation are humble to determine the potential of Acadela. For the expressiveness evaluation, our study could expand to evaluate Acadela's ability to model CPs of other medical fields, e.g., Orthopedics, Cardiovascular, or Acupuncture.

Considering the usability evaluation, learning the opinions from novice software developers can provide further insights into the DSL's user-friendliness. A DSL should be easy-to-use and learnable to new or

potential technical staff joining the industry, thus new modelers can quickly adapt to the tool and capable of modeling CPs for care professionals. Therefore, our study could have explored the usability aspect with newly graduated students possessing qualified degrees, e.g., Computer Science or Bioinformatics.

7.4.7 Internal Validity

Since we did not question possible variations in several CPs, the accuracy of our model reflects the baseline CPs but not their alternatives. Hence, we demonstrated that our CP model could serve as a guideline for specific scenarios, but it only handles a subset of variations. Therefore, we shall evaluate our DSL in the scope of CP variations management to thoroughly examine the expressiveness and applicability of the modeled CP.

7.4.8 External Validity

Acadela leverages the SACM constructs to build CPs, yet other e-Health systems may require certain concepts that do not exist in SACM or Acadela. To further demonstrate applicability, Acadela should generate CPs compatible with other e-Health systems and support the modeling of their domain concepts to provide healthcare services.

8

Conclusion and Future Work

This chapter recapitulates the findings, contributions, reflections, implications, and future work of Acadela, a textual DSL for modeling clinical pathways (CPs) compatible with the Smart Adaptive Case Management (SACM) engine that powers the CONNECARE integrated care system.

8.1 Summary

Research Goal: The author develops Acadela as a textual DSL for constructing generic CPs in e-Health systems which can be interconnected with other services to deliver healthcare operations. The DSL offers an online Integrated Development Environment (IDE) with syntax highlighting and auto-completion to assist modelers in efficiently declaring CP models. Nevertheless, a critical drawback of textual DSL is the absence of visualization to preview the generated CP metamodel. Therefore, Acadela leverages the GoJS tool (Northwoods Software, 2022) to develop a simplified graphical presentation of the CP definition built solely with color codes and common shapes. The motivation of a simple visualization is to reduce the cognitive load of the domain and technical experts in understanding the CP concepts, thus fostering their communication when discussing the accuracy or workflow of the CP.

Implementation: In this study, Acadela leverages the extended Case Management Model and Notation (CMMN) domain concepts of SACM to model unpredictable CPs containing unstructured with pre-defined fragments, structured with ad-hoc exceptions, or structured processes following the Adaptive Case Management (ACM) principles. By leveraging the textX meta-model language, guidelines of DSL design, and error messages, the study implements the grammar, concrete syntax, and constraints validation of Acadela to model CP elements while offering syntax and semantic error detections.

Contribution: Acadela supports modeling concepts to construct essential elements to operate workflows in CPs, along with *communication to external services*, *dynamic graphical representation of data* and the *import* of these elements to foster reusability in CPs. The answer of **RQ1** summarizes the supported modeling concepts in Acadela. Furthermore, Acadela provides a *simplified CP visualization* to deliver an overarching yet intuitive overview of the CP definition code. In this case, the CP definition code serves as the ground truth of the CP model, which modelers declare using the Acadela online Integrated Development Environment (IDE) and syntax. Simultaneously, the visualization displays the process elements defined by the code; thus, double-clicking each graphical component triggers the IDE to focus on the corresponding element declaration code. As a result, modelers can directly read the properties of the elements in textual format and conveniently modify them in the code when applicable.

The below subsections present our findings as answers to the *hypothesis* and four **RQs** in Section 1.2.

Hypothesis

It is possible to define a single DSL which can **model** and **orchestrate** CPs while **fostering communication** between clinical and technical experts.

Acadela is a textual DSL capable of generating SACM-compatible CP meta-models. SACM can create and manage CP models as patient cases from the meta-models. Furthermore, during the CP operation, SACM successfully sends HTTP requests to a defined URL when reaching a process lifecycle. This phenomenon suggests that Acadela can support the inter-service communication required in SACM.

Considering the visualization, in our evaluations, technical and medical professionals state that the graphical notation is understandable. Furthermore, technical professionals regard the visualization as helpful for previewing the generated CP. This fact could imply that technical staff and medical professionals can communicate the workflow accuracy based on graphical presentations.

Our study validates the hypothesis by identifying four RQs, with **RQ1** and **RQ2** explaining the identified modeling concepts and DSL implementation. Meanwhile, **RQ3** and **RQ4** consider the applicability of Acadela by exploring its expressiveness and usability, respectively.

Research Question 1

What elements are required to **model** and **orchestrate** executable Clinical Pathways for Adaptive Case Management?

Through the study of SACM (Michel, 2020) and literature research on existing DSLs for CP modeling, we identified 13 requirements for modeling CP concepts and two supporting features that assist modelers in the modeling process, as described in Section 5.1. Acadela shares the requirements of modeling *workflow phases and activities, control flow, responsibilities, medical document resources, timing constraints* with other DSLs. In addition, Acadela also supports the modeling of three features not supported in existing DSLs: *communication to external services, dynamic graphical representation of data*, and the *import* of CP elements. The three concepts are necessary to model integrated, maintainable, and reusable CPs in modern e-Health systems for the following reasons:

1. E-health systems can apply the microservice architecture or collaborate with external applications to deliver digital healthcare services. CONNECARE is an example of synchronizing monitoring data between the SACM and the Self Management System (SMS). Additionally, an e-Health system may need to consult external services to collect medical data based on certain CP inputs, e.g., retrieve drug-drug interactions between two medicines. Enabling communications in these scenarios requires the CP meta-model to send data to the URLs of the partners' APIs at a particular lifecycle event.
2. CPs may require visualization of medical status to capture an overarching overview of the patient's condition. With dynamic templates, e-Health systems can render adaptive graphical representations based on the value of medical data. Declaring these dynamic templates in the CP definition code enables modelers to create, reuse, and modify the visualization content directly.
3. Various CPs can use the same workflow processes, such as administrative or laboratory test procedures. Therefore, declaring shared processes once and importing them into the required CP fosters reusability and avoid code duplication. Furthermore, modifying the shared processes affects all related CPs at once, thus avoiding cascading updates of the reusable process in different CPs.

Additionally, our study considers two supporting features necessary to ease the modeling process. First, a DSL shall offer an IDE to help modelers declare CP elements. Typically, graphical DSLs require a diagram UI representing graphical notations of CP elements, thus modelers can realize the mapping between visual elements and CP modeling concepts. By assembling and editing the visual elements, modelers can build the CP workflow as required by care professionals. In textual DSLs, the IDE is typically text-based with syntax highlighting to help modelers realize CP concepts or syntactic elements. Auto-completion is also necessary to accelerate the modeling process by creating the structure of any supported CP element from several characters of the keyword.

The second supporting feature is the visualization of CP meta-models. Graphical DSLs UI already supports this feature as modelers see all the CP workflow and control flow during the CP definition. To the best of our knowledge, textual DSLs for CPs typically do not offer the CP visualization feature. Therefore, to compensate the disadvantage of textual DSLs, Acadela outputs a graphical representation of the CP meta-model to provide an overview of the process to the technical and domain experts.

Research Question 2

What are the syntax and semantics that enable a textual DSL to model executable Clinical Pathways for Adaptive Case Management?

Modeling Concepts: Acadela applies the CP domain concepts from SACM to formulate its modeling concepts. However, Acadela groups the schema and behavior definitions of each CP element into a single object, while SACM requires the definitions of a schema, i.e., `EntityDefinitions` and `AttributeDefinitions`, and the behavior in the CP execution, e.g., `StageDefinitions`, `TaskDefinitions`. As a result, the Acadela grammar supports the definitions of `Workspace`, `Case`, `Case Setting`, `Group`, `User`, `Stage`, `Task`, `HttpHook` (external communication), `Precondition` (control flow), `InputField`, and `OutputField`. Each element is a production rule containing all attributes from the schema and behavior definitions of SACM. The advantage of this feature is that modelers declare one CP element, but Acadela compiles the two SACM definitions in the backend to further optimize the CP declaration effort.

Hierarchical Meta-model Definition Structure: Acadela arranges declarations of CP elements in a hierarchal structure. In other words, the CP definition starts with the concept at the highest abstract level, i.e., `Workspace`, which subsumes concepts at the lower level. A `Workspace` contains a `Case`; a `Case` contains `Setting`, `Responsibilities` (`Groups` and `Users`), `Case HttpHooks`, `Summary Panel`, and `Stages`. Each `Stage` comprises `Tasks`, `Preconditions`, and `HttpHooks`. Finally, each `Task` consists of a `Form` that includes `InputFields` or `OutputFields`.

Language Properties: Acadela is a *case-* and *indent-insensitive* language. For each CP element, Acadela expresses user-defined attributes as *key-value pairs*, while attributes with predefined values are in *directive* format, i.e., `#<attributeValue>`. *Directives* leverage the auto-complete feature of the IDE and shorten the statement to declare an attribute.

Syntax Error Analyzer: Acadela can detect *typos*, *unexpected elements*, *unexpected data types*, and *SACM-specific string patterns* (i.e., if-else statements and definition of a color band applied to different ranges of numeric value). The analyzer identifies typos using a dictionary of Acadela keywords and `spellchecker` library. Acadela suggests the most probable keyword by computing the *Levenshtein distance* between the typo and keywords. Regarding the syntax errors of *unexpected elements*, *unexpected data types*, and non-compliant *SACM-specific string patterns*, `textX` grammar provides mechanics to catch the exceptions. However, the error messages can be confusing as they use grammar construct terms to explain the error cause. Therefore, Acadela enhances the error messages by explaining grammar constructs in human-readable modeling concepts.

Semantic Error Analyzer: Acadela has custom semantic analyzers to detect *non-unique identifiers*, *invalid references* to a CP element, and the declaration of *untrusted URL services*. The latter exception is to prevent communications with unidentified parties. The semantic validation starts after analyzing the syntax errors of the CP definition code.

Research Question 3

Can the DSL model Clinical Pathways from **different medical fields** with **diverse complexity** while being **understandable** to **clinical experts**?

Our research evaluates the DSL expressiveness (**RQ3**) by modeling various CPs in SACM and presenting the executable workflow to six medical professionals in five fields of medicine. The result suggests that Acadela can *accurately* model CP workflows in *different medical departments* and *complexity*. Furthermore, medical professionals can identify minor mistakes (e.g., typos) in the CP definition code, implying

References

that Acadela is understandable to domain experts to a certain extent. However, medical professionals wished to see more modeling of visualization and variations in their CP. Additionally, further research as a case study of modeling and applying CPs in medical institutions is necessary to investigate the possibility of fostering communication using the Acadela IDE and visualization.

Research Question 4

Do **modelers** regard the *DSL* and the *development environment* **user-friendly** and **learnable** when modeling CPs?

To evaluate Acadela's usability (**RQ4**), we invited eight technical staff working in the healthcare industry or research to model a simplified CP for hypertension treatment and fix bugs of another CP based on their error messages (EMs). The result implies that Acadela is potentially usable and learnable to technical staff. One reason stated by participants is that Acadela has a "simple" and "easy to learn" syntax. Most participants deem the EMs help them locate, understand and fix syntactic and semantic errors. In addition, they considered the DSL applicable to model CPs in e-health applications. However, we reflected on the result that Acadela is user-friendly given that the modeler has solid basic programming knowledge or above. In addition, the participants wished to see more capabilities, especially the rendering of statistics. Furthermore, the auto-completion should also apply to the CP elements' ID, hence modelers can conveniently insert the element or realize its non-existence. Another concern is that we need more participants to assess the DSL usability substantially.

8.2 Future Work

Evaluate CPs in other Medical Fields: To further validate the scope and practicality of our DSL, one can expand the evaluation for modeling CPs in other medical fields, such as Orthopedics or Traditional Chinese Medicine. In this regard, a longitudinal case study can reveal further drawbacks, hindrances, or advantages of the DSL when modifying changes or incorporating variances in the actual medical environment.

Evaluate Acadela in Different e-Health Systems: Acadela demonstrates that its generated CP is compatible with SACM. However, to become a potential system-independent DSL, Acadela should produce processible CP metamodels in other e-Health systems. Furthermore, compiling an Acadela code to different CP metamodel formats can reveal new challenges. First, can the current generic modeling concepts sufficiently express the CPs of another e-Health system? Second, what adaptations shall apply to the architecture design of the Acadela system, such that it can compile processible CPs in both SACM and other e-Health systems?

References

- Anders, R., Tomai, J., Clute, R., & Olson, T. (1997). Development of a scientifically valid coordinated care path. *The Journal of nursing administration*, 27(5), 45–52.
- Avgar, A., Tambe, P., & Hitt, L. M. (2018). Built to learn: How work practices affect employee learning during healthcare information technology implementation. *Mis Quarterly*, 42(2), 645–660.
- Baar, T. (2015). Verification support for a state-transition-dsl defined with Xtext. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics* (pp. 50–60).
- Bailey, D. A., Litaker, D. G., & Mion, L. C. (1998). Developing Better Critical Paths in Health Care: Combining "Best Practice" and the Quantitative Approach. *Health Care Outcomes: Collaborative, Path-based Approaches*, 70, 34.

References

- Bánáti, A., Kail, E., Karóczkai, K., & Kozlowszky, M. (2018). Authentication and authorization orchestrator for microservice-based software architectures. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1180–1184).
- Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3), 114–123.
- Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., & Parnin, C. (2017). Do developers read compiler error messages? In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* (pp. 575–585).
- Becker, B. A. (2016). An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (p. 126–131). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/2839509.2844584> doi: 10.1145/2839509.2844584
- Becker, B. A., Murray, C., Tao, T., Song, C., McCartney, R., & Sanders, K. (2018). Fix the first, ignore the rest: Dealing with multiple compiler error messages. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 634–639).
- Berdal, K. G., Bøydler, C., Tengs, T., & Holst-Jensen, A. (2008). A statistical approach for evaluation of PCR results to improve the practical limit of quantification (LOQ) of GMO analyses (SIMQUANT). *European Food Research and Technology*, 227(4), 1149–1157.
- Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE cloud computing*, 1(3), 81–84.
- Braun, R., Schlieter, H., Burwitz, M., & Esswein, W. (2014). BPMN4CP: Design and implementation of a BPMN extension for clinical pathways. In *2014 IEEE international conference on bioinformatics and biomedicine (BIBM)* (pp. 9–16).
- Braun, R., Schlieter, H., Burwitz, M., & Esswein, W. (2016). BPMN4CP Revised—Extending BPMN for Multi-perspective Modeling of Clinical Pathways. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (pp. 3249–3258).
- Brown, K. J., Sujeeth, A. K., Lee, H. J., Rompf, T., Chafi, H., Odersky, M., & Olukotun, K. (2011). A heterogeneous parallel framework for domain-specific languages. In *2011 International Conference on Parallel Architectures and Compilation Techniques* (pp. 89–100).
- Brown, S. (n.d.). *The C4 model for visualising software architecture*. Retrieved 2022-11-22, from <https://c4model.com/>
- Büchner, T. (2007). *Introspektive modellgetriebene Softwareentwicklung* (Unpublished doctoral dissertation). Technische Universität München.
- Burns, E. V. (2011). Case management 101: 10 things you must know about case management. *Taming the Unpredictable, Real-world Adaptive Case Management: Case Studies and Practical Guidance*, 17–26.
- Campbell, H., Hotchkiss, R., Bradshaw, N., & Porteous, M. (1998). Integrated care pathways. *Bmj*, 316(7125), 133–137.
- Cano, I., Dueñas-Espín, I., Hernandez, C., de Batlle, J., Benavent, J., Contel, J. C., ... others (2017). Protocol for regional implementation of community-based collaborative management of complex chronic patients. *NPJ primary care respiratory medicine*, 27(1), 1–7.
- Cavusoglu, E. (2021). *A Web-based Breathing Exercise System for Assisting the Treatment of Chronic Obstructive Pulmonary Disease (COPD)* (Doctoral dissertation). Retrieved from <https://www.matthes.in.tum.de/pages/pmdeixilejky/Master-s-Thesis-Elcin-Cavusoglu>
- CDC. (n.d.). *Perceived Exertion (Borg Rating of Perceived Exertion Scale)*. Retrieved 2022-06-17, from <https://www.cdc.gov/physicalactivity/basics/measuring/exertion.htm>

References

- Cook, S., Jones, G., Kent, S., & Wills, A. C. (2007). *Domain-specific development with visual studio dsl tools*. Pearson Education.
- Córdoba-Sánchez, I., & De Lara, J. (2016). Ann: A domain-specific language for the effective design and validation of Java annotations. *Computer Languages, Systems & Structures*, 45, 164–190.
- De Bleser, L., Depreitere, R., Waele, K. D., Vanhaecht, K., Vlayen, J., & Sermeus, W. (2006). Defining pathways. *Journal of nursing management*, 14(7), 553–563.
- De Clercq, J. (2002). Single sign-on architectures. *Infrastructure security*, 40–58.
- Dejanović, I., Milosavljević, G., & Vaderna, R. (2016). Arpeggio: A flexible PEG parser for Python. *Knowledge-based systems*, 95, 71–74.
- Dejanović, I., Vaderna, R., Milosavljević, G., & Vuković, Ž. (2017). TextX: a Python tool for domain-specific languages implementation. *Knowledge-based systems*, 115, 1–4.
- Dejanović, I. (n.d.a). *Multi meta-model support*. Retrieved 2022-07-19, from <https://textx.github.io/textX/3.0/multimetamodel/#use-case-meta-model-referencing-another-meta-model>
- Dejanović, I. (n.d.b). *textx grammar*. Retrieved 2022-12-01, from <https://textx.github.io/textX/3.0/grammar/>
- Dejanović, I. (n.d.c). *Reference resolving expression language (rrel)*. Retrieved 2022-07-21, from <http://textx.github.io/textX/3.0/rrel/>
- Dejanović, I. (n.d.d). *textx meta-models*. Retrieved 2022-12-03, from <https://textx.github.io/textX/3.0/metamodel/>
- Dejanović, I. (n.d.e). *Robot tutorial*. Retrieved 2022-07-19, from <https://textx.github.io/textX/3.0/tutorials/robot/#interpreting-model>
- Di Ciccio, C., Marrella, A., & Russo, A. (2012, 06). Knowledge-intensive processes: An overview of contemporary approaches. In (Vol. 861, p. 33-47).
- Di Ciccio, C., Marrella, A., & Russo, A. (2015). Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics*, 4(1), 29–57.
- Di Lenarda, A., Casolo, G., Gulizia, M. M., Aspromonte, N., Scalvini, S., Mortara, A., ... others (2017). The future of telemedicine for the management of heart failure patients: a Consensus Document of the Italian Association of Hospital Cardiologists (ANMCO), the Italian Society of Cardiology (SIC) and the Italian Society for Telemedicine and eHealth (Digital SIT). *European Heart Journal Supplements*, 19(suppl_D), D113–D129.
- Du, G., Huang, L., & Zhou, M. (2020). Variance Analysis and Handling of Clinical Pathway: An Overview of the State of Knowledge. *IEEE Access*, 8, 158208–158223.
- Every, N. R., Hochman, J., Becker, R., Kopecky, S., & Cannon, C. P. (2000). Critical pathways: a review. *Circulation*, 101(4), 461–465.
- Faber, A. T. L. (2019). *Collaborative modeling and visualizing of business ecosystems* (Dissertation). Technical University Munich, Munich.
- Fender, K. (2018). *EBNF Overview*. Retrieved 2022-12-02, from <https://learn.microsoft.com/en-us/dynamicsax-2012/developer/ebnf-overview>
- Fernández, J. M., Mamei, M., Mariani, S., Felip, M., Alexander, S., Vargiu, E., & Zambonelli, F. (2017). Towards argumentation-based recommendations for personalised patient empowerment. In *2nd International Workshop on Health Recommender Systems*.
- Fernández-Llatas, C., Meneu, T., Benedí, J. M., & Traver, V. (2010). Activity-based Process Mining for Clinical Pathways Computer aided design. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology* (p. 6178-6181). doi: 10.1109/IEMBS.2010.5627760

References

- Ford, B. (2004). Parsing expression grammars: a recognition-based syntactic foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 111–122).
- Frank, U. (2010). *Outline of a method for designing domain-specific modelling languages* (Tech. Rep.). ICB-research report.
- Frank, U. (2011a). *Multi-perspective enterprise modelling: Background and terminological foundation* (Tech. Rep.). ICB-Research Report.
- Frank, U. (2011b). Some guidelines for the conception of domain-specific modelling languages. *Enterprise modelling and information systems architectures (EMISA 2011)*.
- Frank, U. (2013). Domain-specific modeling languages: requirements analysis and design guidelines. In *Domain engineering* (pp. 133–157). Springer.
- Frank, U. (2014). Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. *Software & Systems Modeling*, 13(3), 941–962.
- Fudholi, D. H., & Mutawalli, L. (2018). An ontology model for clinical pathway audit. In *2018 4th International Conference on Science and Technology (ICST)* (pp. 1–6).
- García, M., Clemente, P. E., Moyano, A. F., Pinto, J. M. P., Jimeno, W. L., Urbita, J. A., . . . others (2016). From the Pilot to the Project. Adding remote monitoring into an existing integrated clinical pathway (COMPARTE) to manage COPD and/or HCF patients. *International Journal of Integrated Care (IJIC)*, 16(6).
- Garde, S., & Knaup, P. (2006). Requirements engineering in health care: the example of chemotherapy planning in paediatric oncology. *Requirements Engineering*, 11(4), 265–278.
- Günther, S. (2009). *Agile DSL-Engineering with Patterns in Ruby*. Univ., Fak. für Informatik.
- Hai, J.-J., Wong, C.-K., Un, K.-C., Wong, K.-L., Zhang, Z.-Y., Chan, P.-H., . . . others (2019). Guideline-based critical care pathway improves long-term clinical outcomes in patients with acute coronary syndrome. *Scientific Reports*, 9(1), 1–9.
- Harel, D., & Rumpe, B. (2004). Meaningful modeling: what's the semantics of " semantics"? *Computer*, 37(10), 64–72.
- Hauder, M., Münch, D., Michel, F., Utz, A., & Matthes, F. (2014b). Examining adaptive case management to support processes for enterprise architecture management. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations* (pp. 23–32).
- Hauder, M., Pigat, S., & Matthes, F. (2014a). Research challenges in adaptive case management: a literature review. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations* (pp. 98–107).
- Healey, A. N., Nagpal, K., Moorthy, K., & Vincent, C. A. (2011). Engineering the system of communication for safer surgery. *Cognition, Technology & Work*, 13(1), 1–10.
- Hermans, F., Pinzger, M., & Deursen, A. v. (2009). Domain-specific languages in practice: A user study on the success factors. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 423–437).
- Hernandez-Mendez, A., Michel, F., & Matthes, F. (2018). A practice-proven reference architecture for model-based collaborative information systems. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 13, 262–273.
- Herrmann, C., & Kurz, M. (2011). Adaptive case management: Supporting knowledge intensive processes with it systems. In *International Conference on Subject-Oriented Business Process Management* (pp. 80–97).
- Heß, M. (2013). Towards a domain-specific method for multi-perspective hospital modelling—motivation and requirements. In *International Conference on Design Science Research in Information Systems* (pp. 369–385).

References

- Heß, M., Kaczmarek, M., Frank, U., Podleska, L., & Täger, G. (2015). A domain-specific modelling language for clinical pathways in the realm of multi-perspective hospital modelling.
- Hitt, L. M., & Tambe, P. (2016). Health care information technology, work organization, and nursing home performance. *Ilr Review*, 69(4), 834–859.
- Hollingsworth, D. (2010). Healthcare. In *Keith D. Swenson, editor, Mastering the unpredictable: how adaptive case management will revolutionize the way that knowledge workers get things done* (pp. 163–179). Meghan-Kiffer Press Tampa.
- Horsky, J., Zhang, J., & Patel, V. L. (2005). *Journal of biomedical informatics*, 38(4), 264–266.
- Hyett, K. L., Podosky, M., Santamaria, N., & Ham, J. C. (2007). Valuing variance: the importance of variance analysis in clinical pathways utilisation. *Australian Health Review*, 31(4), 565–570.
- International Organization for Standardization. (2013). *Iso/iec 19510:2013, information technology - object management group business process model and notation*. Retrieved 2022-06-25, from <https://www.iso.org/standard/62652.html>
- John Hopkins Medicine. (n.d.). *Cytology*. Retrieved 2022-06-10, from <https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/cytology>
- Jouault, F., Bézivin, J., & Kurtev, I. (2006). TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering* (pp. 249–254).
- Karpov, A. (2018). *Holy smokes! inhalation injury*. Retrieved 2022-07-10, from <https://www.maimonidesem.org/blog/holy-smokes-inhalation-injury>
- Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., & Völkel, S. (2014). Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378*.
- Kauranen, K., Kim, A., & Osial, P. (2019). Prescriptive Grammar for Clinical Prescribing Workflow. *International Journal of Extreme Automation and Connectivity in Healthcare (IJEACH)*, 1(1), 96–110.
- Khodambashi, S. (2013). Business process re-engineering application in healthcare in a relation to health information systems. *Procedia Technology*, 9, 949–957.
- Kindlmann, G., Chiw, C., Seltzer, N., Samuels, L., & Reppy, J. (2016). Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 22(1), 867–876. doi: 10.1109/TVCG.2015.2467449
- Kinsman, L., Rotter, T., James, E., Snow, P., & Willis, J. (2010). What is a clinical pathway? Development of a definition to inform the debate. *BMC medicine*, 8(1), 1–3.
- Kurz, M. (2013). Taming diversity: A distributed acm-based approach for cross-enterprise knowledge work. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)* (Vol. 3, pp. 87–91).
- Kurz, M., Schmidt, W., Fleischmann, A., & Lederer, M. (2015). Leveraging CMMN for ACM: examining the applicability of a new OMG standard for adaptive case management. In *Proceedings of the 7th international conference on subject-oriented business process management* (pp. 1–9).
- Li, W., Liu, K., Yang, H., & Yu, C. (2014). Integrated clinical pathway management for medical quality improvement—based on a semiotically inspired systems architecture. *European Journal of Information Systems*, 23(4), 400–417.
- Mariani, S., Vargiu, E., Mamei, M., Zambonelli, F., & Miralles, F. (2019). Deliver intelligence to integrate care: the Connecare way. *International Journal of Integrated Care (IJIC)*, 19.

References

- Marin, M. A., Hauder, M., & Matthes, F. (2016). Case management: an evaluation of existing approaches for knowledge-intensive processes. In *International Conference on Business Process Management* (pp. 5–16).
- Matthes, F., Neubert, C., & Steinhoff, A. (2011). Hybrid Wikis: Empowering Users to Collaboratively Structure Information. *ICSOFT (1), 11*, 250–259.
- Matthias, J. T. (2010). Technology for case management. *Mastering the Unpredictable*, 63–88.
- Matthias, J. T. (2011). User requirements for a new generation of case management systems. *Taming the Unpredictable—Real World Adaptive Case Management: Case Studies and Practical Guidance*. Future Strategies Inc., New York.
- McCauley, D. (2011). Acm and business agility for the microsoft-aligned organization. *Taming the Unpredictable: Real World Adaptive Case Management: Case Studies and Practical Guidance*, 65–75.
- Meijer, E., Beckman, B., & Bierman, G. (2006). Linq: reconciling object, relations and xml in the .net framework. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (pp. 706–706).
- Merkle, B. (2010). Textual modeling tools: Overview and comparison of language workbenches. In *Proceedings of the acm international conference companion on object oriented programming systems languages and applications companion* (p. 139–148). New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/1869542.1869564> doi: 10.1145/1869542.1869564
- Michel, F. (2020). *A collaborative purely meta-model-based adaptive case management approach for integrated care* (Dissertation). Technical University Munich, Munich.
- Michel, F., & Matthes, F. (2018). A holistic model-based adaptive case management approach for healthcare. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)* (pp. 17–26).
- Microsoft. (2022). *Monaco editor*. Retrieved 2022-11-24, from <https://microsoft.github.io/monaco-editor/>
- Morgan, R., Grossmann, G., Schrefl, M., Stumptner, M., & Payne, T. (2018). VizDSL: a visual DSL for interactive information visualization. In *International Conference on Advanced Information Systems Engineering* (pp. 440–455).
- Motahari-Nezhad, H. R., & Swenson, K. D. (2013). Adaptive case management: overview and research challenges. In *2013 IEEE 15th conference on business informatics* (pp. 264–269).
- Msoa, Y. J. (2018). *Modelling evolving clinical practice guidelines: a case of Malawi* (Unpublished doctoral dissertation). University of Cape Town.
- Msoa, Y. J. (2019). FCIG grammar evaluation: A usability assessment of clinical guideline modelling constructs. In *2019 IEEE Symposium on Computers and Communications (ISCC)* (pp. 1141–1146).
- Neumann, J., Rockstroh, M., Franke, S., & Neumuth, T. (2016). BPMNSIX—A BPMN 2.0 Surgical Intervention Extension. In *7th workshop on modeling and monitoring of computer assisted interventions (M2CAI), 19th international conference on medical image computing and computer assisted interventions (MICCAI 2016)*, Athens, Greece.
- Neumann, J., Wiemuth, M., Burgert, O., & Neumuth, T. (2017). Application of activity semantics and BPMN 2.0 in the generation and modeling of generic surgical process models. *Journal of the International Foundation for Computer Assisted Radiology and Surgery: International Journal of Computer Assisted Radiology and Surgery*, *12*, 48–49.
- Nielsen, J. (2001). *Error message guidelines*. Retrieved 2022-11-16, from <https://www.nngroup.com/articles/error-message-guidelines/>
- Northwoods Software. (2022). *GoJS - A Web Framework for Rapidly Building Interactive Diagrams*. Retrieved 2022-10-26, from <https://gojs.net/latest/>

References

- Norvig, P. (2022). *pyspellchecker 0.7.1*. Retrieved 2022-12-15, from <https://pypi.org/project/pyspellchecker/>
- Object Management Group. (2015). *Unified modeling language (uml)*. Retrieved 2022-08-19, from <https://www.omg.org/spec/UML/2.5/PDF>
- Object Management Group. (2016). *About the case management model and notation specification version 1.1*. Retrieved 2022-07-13, from <https://www.omg.org/spec/CMMN>
- Object Management Group. (2022). *Case management model and notation™*. Retrieved 2022-07-13, from <https://www.omg.org/cmmn/>
- Oracle. (2022). *Controlling access to members of a class*. Retrieved 2022-07-16, from <https://docs.oracle.com/javase/tutorial/java/java00/accesscontrol.html>
- Panella, M., Marchisio, S., Barbieri, A., & Di Stanislao, F. (2008). A cluster randomized trial to assess the impact of clinical pathways for patients with stroke: rationale and design of the Clinical Pathways for Effective and Appropriate Care Study [NCT00673491]. *BMC health services research*, 8(1), 1–8.
- Panella, M., Marchisio, S., & Di Stanislao, F. (2003). Reducing clinical variations with clinical pathways: do pathways work? *International Journal for Quality in Health Care*, 15(6), 509–521.
- Pearson, S. D., Goulart-Fisher, D., & Lee, T. H. (1995). Critical pathways as a strategy for improving care: problems and potential. *Annals of internal medicine*, 123(12), 941–948.
- Perchetti, G. A., Nalla, A. K., Huang, M.-L., Jerome, K. R., & Greninger, A. L. (2020). Multiplexing primer/probe sets for detection of SARS-CoV-2 by qRT-PCR. *Journal of Clinical Virology*, 129, 104499.
- Preston, S., Markar, S., Baker, C., Soon, Y., Singh, S., & Low, D. (2013). Impact of a multidisciplinary standardized clinical pathway on perioperative outcomes in patients with oesophageal cancer. *Journal of British Surgery*, 100(1), 105–112.
- Reschenhofer, T., & Matthes, F. (2016). Supporting end-users in defining complex queries on evolving and domain-specific data models. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 96–100).
- Rieger, C., Westerkamp, M., & Kuchen, H. (2018). Challenges and Opportunities of Modularizing Textual Domain-Specific Languages. *MODELSWARD*, 387–395.
- Roche. (2016). *A guide for journalist on cervical cancer and its treatment*. Retrieved 2022-07-10, from <https://assets.cwp.roche.com/f/126832/x/92a7b380e2/cervical-cancer-concise-guide-september-2016-pharma.pdf>
- Roche Diagnostics. (2018). *Richtlinie zur gebärmutterhalskrebsvorsorge*. Retrieved 2022-06-10, from https://www.labor-duesseldorf.de/assets/Info-PDF-Sammlung/830e06068e/Abklaerungsalgorithmus_GBA.pdf
- Rotter, T., Kinsman, L., James, E. L., Machotta, A., Gothe, H., Willis, J., ... Kugler, J. (2010). Clinical pathways: effects on professional practice, patient outcomes, length of stay and hospital costs. *Cochrane database of systematic reviews*(3).
- RWTH Aachen Software Engineering Chair. (2022). *Semantics of modeling languages*. Retrieved 2022-12-30, from <https://se-rwth.github.io/research/Semantics/>
- Sandkuhl, K., Stirna, J., Persson, A., & Wißotzki, M. (2014). *Enterprise modeling*. Springer.
- Schlee, M., & Vanderdonckt, J. (2004). Generative programming of graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 403–406).
- Schuld, J., Schäfer, T., Nickel, S., Jacob, P., Schilling, M. K., & Richter, S. (2011). Impact of IT-supported clinical pathways on medical staff satisfaction. A prospective longitudinal cohort study. *International journal of medical informatics*, 80(3), 151–156.

References

- Shen, L., Chen, X., Liu, R., Wang, H., & Ji, G. (2021). Domain-specific language techniques for visual computing: a comprehensive study. *Archives of Computational Methods in Engineering*, 28(4), 3113–3134.
- Shinan, E. (2020). *Grammar reference*. Retrieved 2022-12-02, from <https://lark-parser.readthedocs.io/en/latest/grammar.html>
- Siafis, S., Bursch, N., Müller, K., Schmid, L., Schuster, F., Waibel, J., ... others (2022). Evidence-based Shared-Decision-Making Assistant (SDM-assistant) for choosing antipsychotics: protocol of a cluster-randomized trial in hospitalized patients with schizophrenia. *BMC psychiatry*, 22(1), 1–12.
- Siau, K., & Rossi, M. (2011). Evaluation techniques for systems analysis and design modelling methods—a review and comparative analysis. *Information Systems Journal*, 21(3), 249–268.
- Stroppi, L. J. R., Chiotti, O., & Villarreal, P. D. (2011). Extending BPMN 2.0: method and tool support. In *International Workshop on Business Process Modeling Notation* (pp. 59–73).
- Suduc, A.-M., Bizoi, M., & Filip, F. G. (2010). User awareness about information systems usability. *Studies in Informatics and Control*, 19(2), 145–152.
- Suganthi, S., & Poongodi, T. (2021). Interactive Visualization for Understanding and Analyzing Medical Data. In *Exploratory Data Analytics for Healthcare* (pp. 101–123). CRC Press.
- Swenson, K. (2013). *State of the Art in Case Management. white paper*. Fujitsu.
- Swenson, K., & Palmer, N. (2010). *Mastering the unpredictable: how adaptive case management will revolutionize the way that knowledge workers get things done*. Meghan-Kiffer Press.
- Swenson, K., Palmer, N., & Pucher, M. (2012). Case management: contrasting production vs. adaptive. *How knowledge workers get things done*, 109–118.
- Tongchuan, L., & Deyu, Q. (2013). Dynamic workflow of clinical pathway system. In *Proceedings of the 2012 international conference on communication, electronics and automation engineering* (pp. 109–113).
- Tractinsky, N., Katz, A. S., & Ikar, D. (2000). What is beautiful is usable. *Interacting with computers*, 13(2), 127–145.
- Umrigar, Z. D. (1997). *Introduction to grammars and language analysis*. Retrieved 2022-12-02, from <https://www.cs.binghamton.edu/~zdu/parsdemo/gramintro.html>
- United Kingdom National Health Service. (2021). *Blood pressure test™*. Retrieved 2022-11-03, from <https://www.nhs.uk/conditions/blood-pressure-test/>
- Valentijn, P. P., Schepman, S. M., Opheij, W., & Bruijnzeels, M. A. (2013). Understanding integrated care: a comprehensive conceptual framework based on the integrative functions of primary care. *International journal of integrated care*, 13.
- Van Dam, P. A., Verheyden, G., Sugihara, A., Trinh, X. B., Van Der Mussele, H., Wuyts, H., ... Dirix, L. (2013). A dynamic clinical pathway for the treatment of patients with early breast cancer is a tool for better cancer care: implementation and prospective analysis between 2002–2010. *World journal of surgical oncology*, 11(1), 1–9.
- Van der Aalst, W. M., Weske, M., & Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data & knowledge engineering*, 53(2), 129–162.
- Vanhaecht, K., Bellemans, J., De Witte, K., Diya, L., Lesaffre, E., & Sermeus, W. (2010). Does the organization of care processes affect outcomes in patients undergoing total joint replacement? *Journal of evaluation in clinical practice*, 16(1), 121–128.
- Vanhaecht, K., De Witte, K., Panella, M., & Sermeus, W. (2009). Do pathways lead to better organized care processes? *Journal of evaluation in clinical practice*, 15(5), 782–788.
- Vanhaecht, K., WITTE, K. D., Depreitere, R., & Sermeus, W. (2006). Clinical pathway audit tools: a systematic review. *Journal of nursing management*, 14(7), 529–537.

References

- Vargiu, E., Fernández, J., Miralles, F., Cano, I., Gimeno-Santos, E., Hernandez, C., ... others (2017). Integrated care for complex chronic patients. *International Journal of Integrated Care*, 17(5).
- Vargiu, E., Fernández, J. M., Gonzales-Gonzales, M., Morales-Garzón, J. M., Prunera-Moreda, K., & Miralles, F. (2019). A self-management system for complex chronic patients. *International Journal of Integrated Care (IJIC)*, 19.
- Vargiu, E., Fernández, J. M., & Miralles, F. (2018). Patient empowerment in connecare. *International Journal of Integrated Care (IJIC)*, 18.
- Wang, S., Yu, H., Liu, J., & Liu, B. (2011). Exploring the methodology and application of clinical pathway in evidence-based Chinese medicine. *Frontiers of medicine*, 5(2), 157–162.
- Wang, X., Chen, J., Peng, F., & Lu, J. (2021). Construction of clinical pathway information management system under the guidance of evidence-based medicine. *Journal of Healthcare Engineering*, 2021.
- Wentworth, D. A., & Atkinson, R. P. (1996). Implementation of an acute stroke program decreases hospitalization costs and length of stay. *Stroke*, 27(6), 1040–1043.
- White, M. (2009). Case management: Combining knowledge with process. *BPTrends*, July.
- Wienands, C., & Golm, M. (2009). Anatomy of a visual domain-specific language project in an industrial context. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 453–467).
- Wolff, A. M., Taylor, S. A., & McCabe, J. F. (2004). Using checklists and reminders in clinical pathways to improve hospital inpatient care. *Medical Journal of Australia*, 181(8), 428–431.
- Wooster, L., & Forthman, M. (1996). Establishing a proper perspective on clinical pathways before implementing a clinical improvement program. *Best Practices and Benchmarking in Healthcare: a Practical Journal for Clinical and Management Application*, 1(2), 84–88.
- Yan, H., Van Gorp, P., Kaymak, U., Lu, X., Ji, L., Chiau, C. C., ... Duan, H. (2017). Aligning event logs to task-time matrix clinical pathways in BPMN for variance analysis. *IEEE journal of biomedical and health informatics*, 22(2), 311–317.
- Yang, W., & Su, Q. (2014). Process mining for clinical pathway: Literature review and future directions. In *2014 11th international conference on service systems and service management (ICSSSM)* (pp. 1–5).
- Ye, Y., Jiang, Z., Diao, X., & Du, G. (2009). Knowledge-based hybrid variance handling for patient care workflows based on clinical pathways. In *2009 IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics* (pp. 13–18).
- Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5). sage.
- Zander, K., Bower, K. A., & Etheredge, M. (1987). Nursing case management: blueprints for transformation. *Boston: New England Medical Center Hospitals*, 3.

A

Appendices

A.1 Acadela Complete Grammar

```
1 Start:
2   (versionTag = AcaVersion)?
3   (importList *= Import)?
4
5   (
6     (defWorkspace = DefWorkspace)
7     | ((define)\s/ objList *= Obj)*
8   )
9 ;
10
11 AcaVersion:
12   /(#aca)\d\.\d/
13 ;
14
15 FQN: ID+['.'];
16 FQNI: ID+['.']['.*']?;
17
18 Import:
19   'import' importURI=FQNI ('as' name=ID)?
20 ;
21
22 // Workspace Definition Level
23 DefWorkspace:
24   workspace = Workspace
25   ((define)\s/) case = Case
26 ;
27
28 Workspace:
29   WorkspaceTerm BasicIdentity
30 ;
31
32 BasicIdentity:
33   name = ID
34   ('staticId' Eq staticId=STRING)?
35 ;
36
37 GroupIdentity:
38   name = ID
39   (
40     ('staticId' Eq staticId=STRING)?
41     ('name' Eq groupName=STRING)
42   )#
43 ;
44
45 Obj:
46   ( Case | CaseSetting | Stage | Task | Form | InputField |
47     OutputField | Hook | AttributeValue)
48 ;
```

A Appendices

```
49  /*****
50  ***** CASE *****
51  *****/
52
53  Case:
54  (
55      CaseTerm name=ID
56      (
57          casePrefix = CasePrefix
58          ('version' Eq version = INT)
59          // clientPath and ownerPath is defined in CaseSetting
60          // Interpreter automatically finds the following
61          // attributes:
62          // 1. entityDefinitionId (rootEntityDefinitionId):
63          //    Case Schema: all entities of settings and stages
64          // 2. newEntityAttachPath (entityAttachPath): take path
65          //    to Settings entity by default
66          // 3. newEntityDefinitionId (entityDefinitionId): take
67          //    Settings entity by default
68          //
69          (
70              description = Description
71              responsibilities = Responsibilities
72              setting = CaseSetting
73              summary = SummaryPanel
74              ('Trigger' hookList += CaseHook)?
75              (entityDefinitionId = STRING)?
76              (entityAttachPath = STRING)?
77              (notes = STRING)?
78              stageList += Stage
79          )#
80      )#
81  ) | (Ref /(Case)\s/ ref=[Case|FQN])?
82 ;
83
84 CaseSetting:
85 (
86     SettingTerm
87     (description = Description)?
88     (
89         caseOwner = CaseOwner
90         (attrList *= Attribute)
91         (casePatient = CasePatient)?
92         (attrList *= Attribute)
93     )#
94 ) | (RefSetting ref=[CaseSetting|FQN])?
95 ;
96
97 CaseOwner:
98     /(CaseOwner)\s/ group = TextNoQuote
99     attrProp = AttributeProp
100 ;
101
102 CasePatient:
103     /(CasePatient)\s/ group = TextNoQuote
104     attrProp = AttributeProp
105 ;
106
107 /*LinkUserGroupAttr:
108     directive = AttributeDirective
```

A Appendices

```
107     (
108         //description = Description
109         (attrProp = AttributeProp)?
110     )#
111 ;*/
112
113
114 Responsibilities:
115     /(Responsibilities)\s/
116     (
117         groupList *= Group
118         userList *= User
119     )#
120 ;
121
122 Group:
123     GroupTerm GroupIdentity
124 ;
125
126 User:
127     UserTerm BasicIdentity
128 ;
129
130 AttributeValue:
131     name=ID Eq value=STRING|INT|FLOAT
132 ;
133
134 /*****
135 ***** ATTRIBUTE SECTION *****
136 *****/
137
138 Attribute:
139     AttributeTerm name = ID
140     //(
141
142         attrProp = AttributeProp
143     //)#
144 ;
145
146 AttributeProp:
147     directive = AttributeDirective
148     (
149         description = Description
150         (externalId = ExternalId)?
151         (additionalDescription = AdditionalDescription)?
152         (uiRef = UiReference)?
153         (defaultValue = DefaultValue)?
154         (defaultValues = DefaultValues)?
155     )#
156 ;
157
158 AttributeDirective:
159     (multiplicity = Multiplicity)?
160     (type = Type)?
161 ;
162
163 /*****
164 ***** SUMMARY SECTION *****
165 *****/
166
167 SummaryPanel:
```

A Appendices

```
168     /(SummaryPanel)\s/ sectionList += SummarySection
169 ;
170
171 SummarySection:
172     /(Section)\s/ name = ID
173     (directive = SummarySectionPosition)?
174     description = Description
175     paramList += SummaryParam
176 ;
177
178 SummaryParam:
179     /(InfoPath)\s/ path = TextNoQuote
180 ;
181
182 /*****
183 ***** HTTPHOOK *****
184 *****/
185
186 Hook:
187     CaseHook | HttpHook | DualTaskHttpHook
188 ;
189
190 CaseHook:
191     (
192         /(Hook)\s/ name = ID)?
193         /(On)\s/ event = CaseHookEvent
194         /(invoke)\s/ url = STRING
195     ) | (RefHook ref=[CaseHook|FQN])?
196 ;
197
198 HttpHook:
199     (
200         /(Hook)\s/ name = ID)?
201         /(On)\s/ event = BaseEvent
202         (
203             /(invoke)\s/ url = STRING)
204             /(method)\s/ method = HttpMethod)
205         /(with failureMessage)\s/ failureMessage = STRING)?
206     )#
207     ) | (RefHook ref=[HttpHook|FQN])?
208 ;
209
210 DualTaskHttpHook:
211     (
212         /(Hook)\s/ name = ID)?
213         /(On)\s/ event = DualTaskEvent
214         (
215             /(invoke)\s/ url = STRING)
216             /(method)\s/ method = HttpMethod)
217         /(with failureMessage)\s/ failureMessage = STRING)?
218     )#
219     ) | (RefHook ref=[DualTaskHttpHook|FQN])?
220 ;
221
222 SharedEvent:
223     'available'
224     | 'enable'
225     | 'activate'
226     | 'complete'
227     | 'terminate'
228 ;
```

A Appendices

```
229
230 CaseHookEvent:
231     SharedEvent
232     | 'delete'
233 ;
234
235 BaseEvent:
236     SharedEvent
237     | 'correct'
238 ;
239
240 DualTaskEvent:
241
242     'activatehumanpart'
243     | 'activateautopart'
244     | 'completehumanpart'
245     | 'completeautopart'
246     | 'correcthumanpart'
247     | 'correctautopart'
248     | BaseEvent
249 ;
250
251 CasePrefix:
252     'prefix' Eq value = STRING
253 ;
254
255 Description:
256     ('label' Eq value = STRING)
257 ;
258
259
260 /******
261 ***** STAGE *****
262 *****/
263
264 Stage:
265 (
266     StageTerm name = ID
267     directive = WorkflowElementDirective
268     (
269         (description = Description)
270         (ownerPath = OwnerPath)?
271         (clientPath = ClientPath)?
272         (dynamicDescriptionPath = DynamicDescriptionPath)?
273         (externalId = ExternalId)?
274         (additionalDescription = AdditionalDescription)?
275     )#
276     (
277         (preconditionList *= Precondition)
278         (TriggerTerm hookList *= HttpHook)?
279         taskList += Task
280     )#
281 ) | (RefStage ref=[Stage|FQN])?
282 ;
283
284 /******
285 ***** TASK *****
286 *****/
287
288 Task:
289     HumanTask
```


A Appendices

```
290 | AutomatedTask
291 | DualTask
292 ;
293
294 HumanTask:
295 (
296     HumanTaskTerm name = ID
297     directive = WorkflowElementDirective
298     attrList = SharedTaskAttrs
299     (
300         (TriggerTerm hookList *= HttpHook)?
301         form = Form
302     )#
303 )
304 | (RefTask ref=[HumanTask|FQN])?
305 ;
306
307 AutomatedTask:
308 (
309     AutoTaskTerm name = ID
310     directive = WorkflowElementDirective
311     attrList = AutomatedTaskAttrs
312     (
313         (TriggerTerm hookList *= HttpHook)?
314         form = Form
315     )#
316 ) | (RefTask ref=[AutomatedTask|FQN])?
317 ;
318
319 DualTask:
320 (
321     DualTaskTerm name = ID
322     directive = WorkflowElementDirective
323     attrList = SharedTaskAttrs
324     (
325         (TriggerTerm hookList *= DualTaskHttpHook)?
326         form = Form
327     )#
328 ) | (RefTask ref=[DualTask|FQN])?
329 ;
330
331 AutomatedTaskAttrs:
332 (
333     description = Description
334     (ownerPath = OwnerPath)?
335     (externalId = ExternalId)?
336     (dynamicDescriptionPath = DynamicDescriptionPath)?
337     (additionalDescription = AdditionalDescription)?
338     (preconditionList *= Precondition)
339 )#
340 ;
341
342 SharedTaskAttrs:
343 (
344     description = Description
345     (ownerPath = OwnerPath)?
346     (dueDatePath = DueDatePath)?
347     (externalId = ExternalId)?
348     (additionalDescription = AdditionalDescription)?
349     (dynamicDescriptionPath = DynamicDescriptionPath)?
350     (preconditionList *= Precondition)
```

A Appendices

```
351     )#
352 ;
353
354 Precondition:
355     PreconditionTerm
356     (
357         // aka. processDefinitionId in Thesis
358         ('previousStep' Eq stepList += STRING)*
359
360         // aka. expression in Thesis
361         ('condition' Eq entryCondition = STRING)?
362     )#
363 ;
364
365 /*****
366 ***** FORM *****
367 *****/
368
369 Form:
370     (
371         FormTerm name = ID
372             (directive = FormDirective)?
373             fieldList += FormField
374     ) | (RefForm ref=[Form|FQN])?
375 ;
376
377 FormDirective:
378     (
379         (mandatory = Mandatory)?
380         (readOnly = ReadOnly)?
381     )#
382 ;
383
384 FormField:
385     InputField/[\s\n]*/ | OutputField/[\s\n]*/
386 ;
387
388 InputField:
389     (
390         InputFieldTerm name = ID
391         directive = InputFieldDirective
392         (
393             (
394                 description = Description
395                 | question = Question
396             )
397             (path = CustomElementRefPath)?
398             (uiRef = UiReference)?
399             (externalId = ExternalId)?
400             (additionalDescription = AdditionalDescription)?
401             (defaultValue = DefaultValue)?
402             (defaultValues = DefaultValues)?
403         )#
404     ) | (Ref InputFieldTerm ref=[InputField|FQN])?
405 ;
406
407 CustomElementRefPath:
408     'ElementPath' Eq value=STRING
409 ;
410
411 InputFieldDirective:
```

A Appendices

```

412 (
413     (mandatory = Mandatory)?
414     (readOnly = ReadOnly)?
415     (position = Position)?
416     (multiplicity = Multiplicity)?
417     // There is no grammar for DualTask field
418     // to avoid overhead in computation, since
419     // we expect a large number of fields, and
420     // introducing a DualTaskField along with InputField
421     // will result in longer parsing time
422     // The interpreter will check whether a part
423     // is included in the DualTaskField instead.
424     (part = Part)?
425     (type = FieldType)?
426 )#
427 ;
428
429 OutputField:
430 (
431     OutputFieldTerm name = ID
432     directive = OutputFieldDirective
433     (
434         description = Description
435         (additionalDescription = AdditionalDescription)?
436         (uiRef = UiReference)?
437         (path = CustomElementRefPath)?
438         (expression = OutputFieldExpression)?
439         (externalId = ExternalId)?
440         (defaultValue = DefaultValue)?
441         (defaultValues = DefaultValues)?
442     )#
443 ) | (Ref OutputFieldTerm ref=[OutputField|FQN])?
444 ;
445
446 OutputFieldDirective:
447 (
448     (mandatory = Mandatory)?
449     (readOnly = ReadOnly)?
450     (position = Position)?
451     //(explicitType = PrimitiveDataType)?
452     (explicitType = Type)?
453 )#
454 ;
455
456 Question:
457 'Question' Eq text=STRING
458 optionList += Option
459 ;
460
461 Option:
462 /(Option)\s/ (
463     (key=STRING)
464     ('value' Eq value=STRING)
465     (additionalDescription = AdditionalDescription)?
466     (externalId = ExternalId)?
467 )#
468 ;
469
470 /*****
471 ***** COMMON DIRECTIVES *****
472 *****/

```

A Appendices

```
473
474 WorkflowElementDirective:
475     (
476         (mandatory = Mandatory)?
477         (repeatable = Repeatable)?
478         (activation = Activation)?
479         (multiplicity = Multiplicity)?
480     )#
481 ;
482
483 Multiplicity:
484     Hash (
485         'maxOne'
486         | 'exactlyOne'
487         | 'atLeastOne'
488         | 'any'
489     )
490 ;
491
492 Type:
493     Hash (
494         LinkType
495         | DocumentLinkType
496         | 'notype'
497         | 'text' // string
498         | 'longtext'
499         | 'string'
500         | 'boolean'
501         | NumType
502         | 'singlechoice' // aka. 'enumeration' in Thesis
503         | 'multiplechoice'
504         | DateType
505         | 'json'
506         | 'custom'
507     )
508 ;
509
510 FieldType:
511     Type
512 ;
513
514 LinkType:
515     'link' '.' (linkType='Users' | linkType='Entity')
516     '(' linkObj += TextNoQuote (',' linkObj += TextNoQuote)? ')'
517 ;
518
519 DocumentLinkType:
520     'documentlink' '(' url=STRING ')'
521 ;
522
523 DateType:
524     'date.after(TODAY)'
525     | /(date)\s/
526 ;
527
528 NumType:
529     'number' '(' (
530         ((comparator=Comparator num=INT) | (min=INT '-' max=INT))
531     )')'?
532 ;
533
```

A Appendices

```
534 Part:
535     Hash (
536         'humanDuty'
537         | 'systemDuty'
538     )
539 ;
540
541 Repeatable:
542     Hash (
543         'repeatSerial'
544         | 'repeatParallel' ( '(' INT ')' )?
545         | 'noRepeat' // default
546     )
547 ;
548
549 Mandatory:
550     Hash (
551         'mandatory' // default
552         | 'notmandatory'
553     )
554 ;
555
556 Activation:
557     Hash (
558         'manualActivate'
559         | 'autoActivate' // default
560         | 'activateWhen' '(' STRING ')',
561     )
562 ;
563
564 ReadOnly:
565     Hash (
566         'readOnly'
567         | 'notReadOnly' // default
568     )
569 ;
570
571 SharedPosition:
572     'stretched' // Default
573     | 'left'
574     | 'center'
575     | 'right'
576 ;
577
578 SummarySectionPosition:
579     Hash SharedPosition
580 ;
581
582 Position:
583     Hash (
584         'leftcenter'
585         | 'centerright'
586         | SharedPosition
587     )
588 ;
589
590
591 /*****
592 ***** SHARED PROPS *****
593 *****/
594
```

A Appendices

```
595 AdditionalDescription:
596     "additionalDescription" Eq value=STRING
597 ;
598
599 DefaultValues:
600     'defaultValues' Eq value=WrapValue
601 ;
602
603 DefaultValue:
604     (
605         'defaultValue' Eq
606         (
607             (value=STRING)
608             | (Ref ref=[AttributeValue|FQN])
609         )
610     )
611 ;
612
613 DynamicDescriptionPath:
614     'dynamicDescriptionRef' Eq value=STRING
615 ;
616
617 ExternalId:
618     'externalId' Eq value = STRING
619 ;
620
621 OwnerPath:
622     'owner' Eq value = STRING
623 ;
624
625 ClientPath:
626     'client' Eq value = STRING
627 ;
628
629 UiReference:
630     'uiRef' Eq
631     (
632         ( value=STRING )
633         | (Ref ref=[AttributeValue|FQN])
634     )
635 ;
636
637 DueDatePath:
638     'dueDateRef' Eq value=STRING
639 ;
640
641 OutputFieldExpression:
642     'expression' Eq value = STRING
643 ;
644
645 /*****
646 ***** TERMINAL *****
647 *****/
648
649 HttpMethod:
650     'get'
651     | 'post'
652     | 'put'
653     | 'delete'
654 ;
655
```

A Appendices

```
656 Comparator:
657     '='
658     | '<>'
659     | '<='
660     | '>='
661     | '<'
662     | '>'
663 ;
664
665 Quote:
666     '"' | "'"
667 ;
668
669 Text:
670     STRING
671 ;
672
673 TextNoQuote:
674     /([a-zA-Z0-9-_.]）*/
675 ;
676
677 // Assignment Sign
678 Eq:
679     '='
680 ;
681
682 // Directive Sign
683 Hash:
684     '#'
685 ;
686
687 WrapValue:
688     '[' (STRING | TextNoQuote) (',' (STRING | TextNoQuote))* ']'
689 ;
690
691 Ref:
692     /(use)\s/
693 ;
694
695 RefSetting:
696     Ref SettingTerm
697 ;
698
699 RefStage:
700     Ref StageTerm
701 ;
702
703 RefTask:
704     Ref TaskTerm
705 ;
706
707 RefField:
708     Ref InputFieldTerm
709 ;
710
711 RefForm:
712     Ref FormTerm
713 ;
714
715 RefHook:
716     Ref HookTerm
```

A Appendices

```
717 ;
718
719 Comment :
720     (/\//.*$/
721     | /\//.*$/
722     | /\//.*$/
723     | /\//.*$/ )
724 ;
725
726 /*****
727 ***** I18N TERMINAL *****
728 *****/
729
730 WorkspaceTerm :
731     /(Workspace)\s/
732 ;
733
734 CaseTerm :
735     /(Case)\s/
736 ;
737
738 SettingTerm :
739     /(Setting)\s/
740 ;
741
742 StageTerm :
743     /(Stage)\s/
744 ;
745
746 TaskTerm :
747     /(Task)\s/
748 ;
749
750 HumanTaskTerm :
751     /(HumanTask)\s/
752 ;
753
754 AutoTaskTerm :
755     /(AutoTask)\s/
756 ;
757
758 DualTaskTerm :
759     /(DualTask)\s/
760 ;
761
762 FormTerm :
763     /(Form)\s/
764 ;
765
766 InputFieldTerm :
767     /(InputField)\s/
768 ;
769
770 OutputFieldTerm :
771     /(OutputField)\s/
772 ;
773
774 TriggerTerm :
775     /(Trigger)\s/
776 ;
777
```


A Appendices

```
778 HookTerm:
779     /(Hook)\s/
780 ;
781
782 UserTerm:
783     /(User)\s/
784 ;
785
786 GroupTerm:
787     /(Group)\s/
788 ;
789
790 PreconditionTerm:
791     /(Precondition)\s/
792 ;
793
794 FormTerm:
795     /(Form)\s/
796 ;
797 AttributeTerm:
798     /(Attribute)\s/
799 ;
800
801 /*****
802 **** ERROR DETECTION ASSISTANT RULES ****
803 *****/
804
805 // These are the foundational object in SACM. When the
806 // Syntax Detector scan the lines upward from the error line,
807 // if it reaches one of these keywords, it will stop scanning
808
809 BuildingBlockObject:
810     WorkspaceTerm
811     | CaseTerm
812     | SettingTerm
813     | StageTerm
814     | HumanTaskTerm | DualTaskTerm | AutoTaskTerm
815     | FormTerm
816     | InputFieldTerm | OutputFieldTerm
817     | GroupTerm | UserTerm
818     | PreconditionTerm | HookTerm
819 ;
```

Listing A.1: The complete Acadela grammar specification.

A.2 CP Model Definition and Visualization in Acalada

This section presents the Acalada code and visualization of the six CPs presented in Section 7.2.2. The groups and CP ID in the workspace is anonymized to avoid exposing the identity of the medical institutions.

A.2.1 COPD Breathing Exercise

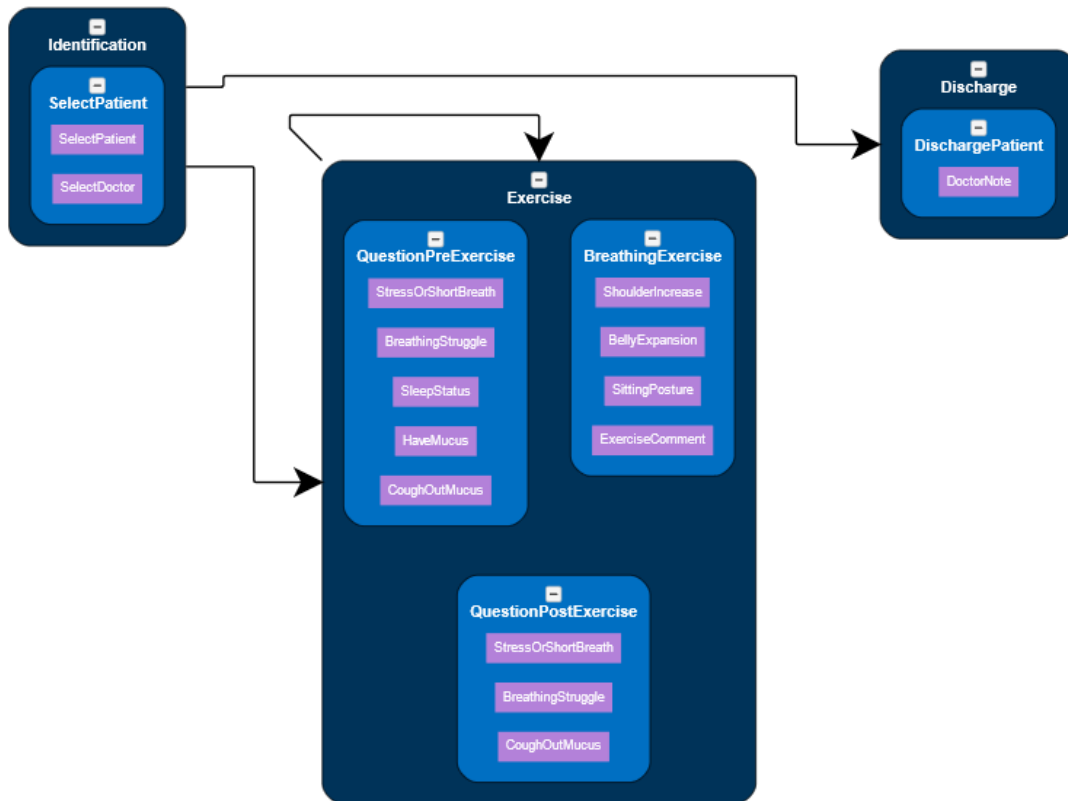


Figure A.1: Visualization of the CP model generated by Listing A.2.

```

1  aca0.1
2  workspace Demo
3  define case PII1_COPD
4    prefix = 'PII1'
5    version = 1
6    label = 'COPD Treatment'
7
8  Responsibilities
9    group DemoPhysicians name = 'Demo Physician' //staticId = '
asdf234'
10   group DemoClinicians name = 'Demo Clinician'
11   group DemoProfessionals name = 'Demo Professional'
12   group DemoPatients name = 'Demo Patient'
13   group DemoNurses name = 'Demo Nurse'
14
15  Setting
16   CaseOwner DemoProfessionals
17   #exactlyOne
18   label = 'Demo Professionals'

```

A Appendices

```
20 Attribute WorkplanDueDate
21   #exactlyOne #date.#after(TODAY)
22   label = 'Workplan Due Date'
23   externalId = 'dueDateConnie'

25 CasePatient DemoPatients
26   #exactlyOne
27   label = 'Patient'

29 Attribute Clinician
30   #exactlyOne #Link.#Users(DemoClinicians)
31   label = 'Clinician'

33 Attribute Nurse
34   #exactlyOne #Link.#Users(DemoNurses)
35   label = 'Nurse'

37 SummaryPanel
38   Section PrevExerciseShoulder #left
39     label = "Shoulder was increased accurately?"
40     InfoPath Exercise.BreathingExercise.ShoulderIncrease

42   Section PrevBellyExpansion #left
43     label = "Belly was expanded properly?"
44     InfoPath Exercise.BreathingExercise.BellyExpansion

46   Section PrevSittingPosture #left
47     label = "Sitting posture was correct?"
48     InfoPath Exercise.BreathingExercise.SittingPosture

50   Section DoctorNote #left
51     label = "Recommendations"
52     InfoPath Discharge.DischargePatient.DoctorNote

54 Stage Identification
55   #mandatory
56   owner = 'Setting.CaseOwner'
57   label = 'Identification'

59 HumanTask SelectPatient
60   #mandatory
61   label = 'Assign Patient'
62   owner = 'Setting.Nurse'
63   dueDateRef = 'Setting.WorkplanDueDate'
64   externalId = 'SelectPatient'

66 Form PatientAssignForm
67   #mandatory

69   InputField SelectPatient
70     #custom
71     ElementPath = "Setting.CasePatient"
72     label = "Assigned Patient"

74   InputField SelectDoctor
75     #custom
76     ElementPath = "Setting.Clinician"
77     label = "Assigned Clinician"

79 Stage Exercise
80   #mandatory #repeatSerial #atLeastOne
```

A Appendices

```
81     label = "Exercise"
83
84     Precondition
85         previousStep = 'Identification'
86
87     Precondition
88         previousStep = 'Exercise'
89
90     HumanTask QuestionPreExercise
91         #mandatory #atLeastOne
92         label = "Pre-exercise Questionnaire"
93
94     Form PreExerciseForm
95         #mandatory
96         InputField StressOrShortBreath
97             #singlechoice #stretched
98             question = 'Do you feel stressed or short of breath today
99             ?'
100                 option 'Not at all' value = '4'
101                 option 'A little' value = '3'
102                 option 'So-so' value = '2'
103                 option 'Yes' value = '1'
104                 option 'Too much' value = '0'
105                 uiRef = "colors(0<=red<=1<yellow<=2<=green<=4)"
106
107     InputField BreathingStruggle
108         #singlechoice #stretched
109         question = 'Do you struggle to breath?'
110             option 'Not at all' value = '4'
111             option 'A little' value = '3'
112             option 'So-so' value = '2'
113             option 'Yes' value = '1'
114             option 'Too much' value = '0'
115             uiRef = 'colors(0<=red<=1<yellow<=2<=green<=4)'
116
117     InputField SleepStatus
118         #singlechoice #stretched
119         question = 'How was your sleep last night?'
120             option 'Good' value = '2'
121             option 'Medium' value = '1'
122             option 'Bad' value = '0'
123             uiRef = 'colors(0<=red<1<=yellow<2<=green<=10)'
124
125     InputField HaveMucus
126         #left #singlechoice
127         question = 'Do you have mucus today?'
128             option 'No' value = '0'
129             option 'Yes' value = '1'
130
131     InputField CoughOutMucus
132         #left #singlechoice
133         question = 'Can you cough your mucus out?'
134             option 'No' value = '0'
135             option 'Yes' value = '1'
136
137     HumanTask BreathingExercise
138         #mandatory #atLeastOne
139         label = 'Conduct Breathing Exercise'
140
141     precondition
```

A Appendices

```
141         previousStep = 'QuestionPreExercise'
143     Form ExerciseEvalForm
144         #mandatory
145         InputField ShoulderIncrease
146             #left #singlechoice
147             question = 'The shoulder position is accurate.'
148             option 'No' value = '0'
149             option 'Almost' value = '1'
150             option 'Yes' value = '2'
152
153         InputField BellyExpansion
154             #left #singlechoice
155             question = 'The belly contraction and expansion is
accurate.'
156             option 'No' value = '0'
157             option 'Almost' value = '1'
158             option 'Yes' value = '2'
159
160         InputField SittingPosture
161             #stretched #singlechoice
162             question = 'The sitting posture of the patient is
accurate.'
163             option 'No' value = '0'
164             option 'Almost' value = '1'
165             option 'Yes' value = '2'
166             option 'Not Applicable' value = '-1'
167
168         InputField ExerciseComment
169             #stretched #notmandatory
170             label = 'Comment'
171
172     HumanTask QuestionPostExercise
173         #mandatory #atLeastOne
174         label = 'Post-exercise Questionnaire'
175
176         precondition
177             previousStep = 'BreathingExercise'
178
179     Form PostExerciseForm
180         #mandatory
181         InputField StressOrShortBreath
182             #singlechoice #stretched
183             question = 'Do you feel stressed or short of breath today
?'
184             option 'Not at all' value = '4'
185             option 'A little' value = '3'
186             option 'So-so' value = '2'
187             option 'Yes' value = '1'
188             option 'Too much' value = '0'
189             uiRef = 'colors(0<=red<=1<yellow<=2<=green<=4)'
190
191         InputField BreathingStruggle
192             #singlechoice #stretched
193             question = 'Do you struggle to breath?'
194             option 'Not at all' value = '4'
195             option 'A little' value = '3'
196             option 'So-so' value = '2'
197             option 'Yes' value = '1'
198             option 'Too much' value = '0'
```

A Appendices

```

199         uiRef = 'colors(0<=red<=1<yellow<=2<=green<=4)'
201     InputField CoughOutMucus
202         #left #singlechoice
203         question = 'Can you cough your mucus out?'
204         option 'No' value = '0'
205         option 'Yes' value = '1'

208     Stage Discharge
209         #mandatory #manualActivate
210         owner = 'Setting.CaseOwner'
211         label = 'Discharge'

213     precondition
214         previousStep = 'Identification'

216     HumanTask DischargePatient
217         #mandatory
218         owner = 'Setting.CaseOwner'
219         label = 'Discharge Patient'

221     Form DischargeForm
222         InputField DoctorNote
223         #text
224         label = 'Post-Treatment Recommendation:'

```

Listing A.2: Acadela code to define the CP of COPD Breathing Exercise

A.2.2 Selection of Antipsychotics for Schizophrenia

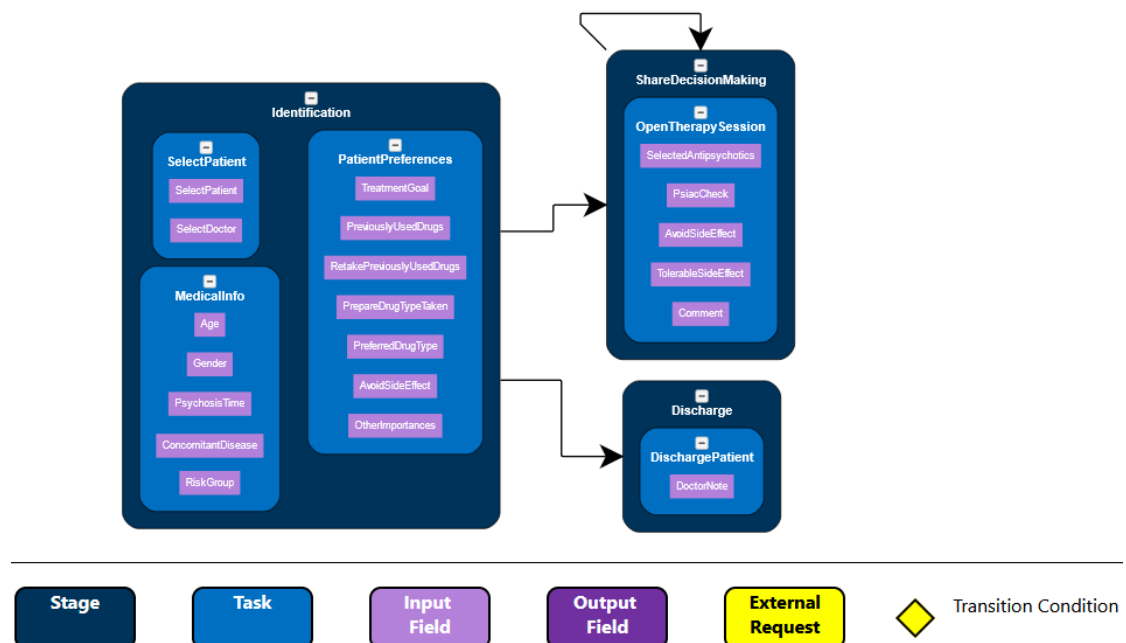


Figure A.2: Visualization of the CP model generated by Listing A.3.

A Appendices

```
1  aca0.1
2  workspace Demo

4  define case STP_Schizophrenia
5      prefix = 'STP'
6      version = 2
7      label = 'Schizophrenia Treatment'

9  Responsibilities
10     group DemoPhysicians name = 'Demo Physician' //staticId = 'asdf234'
11     group DemoClinicians name = 'Demo Clinician'
12     group DemoProfessionals name = 'Demo Professional'
13     group DemoPatients name = 'Demo Patient'
14     group DemoNurses name = 'Demo Nurse'

16     // A comment
17     /* a multiline
18        * Comment
19        */

21     Setting
22         // label = "Case Configuration"
23         CaseOwner DemoProfessionals #exactlyOne
24             label = 'Demo Professionals'

26         Attribute WorkplanDueDate
27             #exactlyOne #date.#after(TODAY)
28             label = 'Workplan Due Date'
29             externalId = 'dueDateConnie'

31         CasePatient DemoPatients #exactlyOne
32             label = 'Patient'

34         Attribute Clinician
35             #exactlyOne #Link.#Users(DemoClinicians)
36             label = 'Clinician'

38         Attribute Nurse
39             #exactlyOne #Link.#Users(DemoNurses)
40             label = 'Nurse'

42     SummaryPanel
43         Section MedicalInformation #stretched
44             label = "Medical Information:"
45             InfoPath Identification.MedicalInfo.Age
46             InfoPath Identification.MedicalInfo.Gender
47             InfoPath Identification.MedicalInfo.PsychosisTime
48             InfoPath Identification.MedicalInfo.ConcomitantDisease
49             InfoPath Identification.MedicalInfo.RiskGroup

51         Section PatientPreferences #stretched
52             label = "Patient Preferences:"
53             InfoPath Identification.PatientPreferences.TreatmentGoal
54             InfoPath Identification.PatientPreferences.PreviouslyUsedDrugs
55             InfoPath Identification.PatientPreferences.AvoidSideEffect
56             InfoPath Identification.PatientPreferences.PreferredDrugType
57             InfoPath Identification.PatientPreferences.OtherImportances

59         Section LastTherapySession #stretched
60             label = "Patient Preferences:"
```

A Appendices

```
61     InfoPath ShareDecisionMaking.OpenTherapySession.
SelectedAntipsychotics
62     InfoPath ShareDecisionMaking.OpenTherapySession.AvoidSideEffect
63     InfoPath ShareDecisionMaking.OpenTherapySession.
TolerableSideEffect

64
65 Stage Identification
66 #mandatory
67 owner = 'Setting.CaseOwner'
68 label = 'Identification'

69
70 HumanTask SelectPatient
71 #mandatory
72 label = 'Assign Patient'
73 owner = 'Setting.Nurse'
74 dueDateRef = 'Setting.WorkplanDueDate'
75 externalId = 'SelectPatient'

76
77 Form PatientAssignForm
78 #mandatory

79
80 InputField SelectPatient
81 #custom
82 ElementPath = "Setting.CasePatient"
83 label = "Assigned Patient"

84
85 InputField SelectDoctor
86 #custom
87 ElementPath = "Setting.Clinician"
88 label = "Assigned Clinician"

89
90 HumanTask MedicalInfo
91 #mandatory
92 label = "Medical Information"
93 Form MedicalInfoForm
94 #mandatory
95 InputField Age
96 #number
97 label = 'Age:'

98
99 InputField Gender
100 #singlechoice
101 question = 'Gender:'
102 option 'Male' value = '0'
103 option 'Female' value = '1'
104 option 'Other' value = '2'

105
106 InputField PsychosisTime
107 #number
108 label = 'Length of Psychosis (Months):'

109
110 InputField ConcomitantDisease
111 #multipleChoice #left
112 additionalDescription = 'Is any of the following concomitant
diseases known to you?'
113 question = 'Known Concomitant Disease:'
114 Option "Cardiac Diseases" value = "1"
115 Option "Epilepsy" value = "2"
116 Option "Liver Diseases" value = "3"
117 Option "Kidney Diseases" value = "4"
118 Option "Adipositas" value = "5"
```


A Appendices

```
119         Option "Diabetes" value = "6"
120         Option "Fat Metabolism Disorder" value = "7"
121         Option "Blood Count Changes" value = "8"
122         Option "Cognitive Changes/Dementias" value = "9"

124     InputField RiskGroup
125         #multipleChoice #left
126         question = 'Which sub-group does the patient belongs to?'
127         option 'Adolescence' value = '1'
128         option 'Senior' value = '2'
129         option 'Comorbid Substance Abuse' value = '3'
130         option 'Predominantly Negative Symptoms' value = '4'
131         option 'Psychologically Stable Patient' value = '5'
132         option 'Therapy resistance' value = '6'
133         option 'Pregnant' value = '7'

135     HumanTask PatientPreferences
136         #mandatory
137         label = 'Record Medical Profile'

139     Form PrefForm
140         #mandatory

142     InputField TreatmentGoal
143         #text
144         label = 'What would I like to achieve after the treatment? (
Treatment Goal):'

146     InputField PreviouslyUsedDrugs
147         #singlechoice #left #atLeastOne
148         Question = 'Previously Used Antipsychotics:'
149         Option "Amisulprid" value = "1"
150         Option "Aripirazol" value = "2"
151         Option "Cariprazin" value = "3"
152         Option "Clozapin" value = "4"
153         Option "Haloperidol" value = "5"
154         Option "Olanzapin" value = "6"
155         Option "Paliperidon" value = "7"
156         Option "Risperidon" value = "8"
157         Option "Perphenazin" value = "9"
158         Option "Quetiapin" value = "10"
159         Option "Sertindol" value = "11"
160         Option "Ziprasidon" value = "12"

163     InputField RetakePreviouslyUsedDrugs
164         #multipleChoice #center
165         Question = 'Which Drugs would be Used again:'
166         Option "Amisulprid" value = "Amisulprid"
167         Option "Aripirazol" value = "Aripirazol"
168         Option "Cariprazin" value = "Cariprazin"
169         Option "Clozapin" value = "Clozapin"
170         Option "Haloperidol" value = "Haloperidol"
171         Option "Olanzapin" value = "Olanzapin"
172         Option "Paliperidon" value = "Paliperidon"
173         Option "Risperidon" value = "Risperidon"
174         Option "Perphenazin" value = "Perphenazin"
175         Option "Quetiapin" value = "Quetiapin"
176         Option "Sertindol" value = "Sertindol"
177         Option "Ziprasidon" value = "Ziprasidon"
```

A Appendices

```
179     InputField PrepareDrugTypeTaken
180         #singlechoice #stretched
181         Question = 'Medications can be administered as tablets, drops
, or injections (usually several weeks apart). Do you want to start
thinking about this now?'
182         Option "Yes" value = "1"
183         Option "No" value = "0"

185     InputField PreferredDrugType
186         #singlechoice #left #atLeastOne
187         Question = 'Preferred Drug Types:'
188         Option "Syringes" value = "1"
189         Option "Pills" value = "2"
190         Option "Drops" value = "3"

192     InputField AvoidSideEffect
193         #stretched #multipleChoice
194         Question = 'Side Effects to be Avoided:'
195         Option 'Dry Mouth, Blurred Vision, Constipation' value = '1
,
196         Option 'Muscular Stiffness, Movement Disorders, Tremor'
value = '2'
197         Option 'Reduction of Sexual Desire, Sexual Dysfunction,
Menstrual Cramps' value = '3'
198         Option 'Weight Gain' value = '4'
199         Option 'Fatigue' value = '5'
200         Option 'Restless Legs' value = '6'

202     InputField OtherImportances
203         #text #notmandatory
204         label = 'Other Important Notes:'

206 Stage ShareDecisionMaking
207     #mandatory #repeatSerial
208     owner = 'Setting.Clinician'
209     label = 'Share Decision Making'

211 Precondition
212     previousStep = 'Identification'

214 Precondition
215     previousStep = 'ShareDecisionMaking'

217 HumanTask OpenTherapySession
218     #mandatory #repeatParallel #atLeastOne
219     label = 'Arrange Therapy Session'
220     owner = 'Setting.Clinician'
221     dueDateRef = 'Setting.WorkplanDueDate'

223 Form SDMForm
224     #mandatory
225     InputField SelectedAntipsychotics
226         #multipleChoice
227         Question = 'Selected Antipsychotics:'
228         Option "Amisulprid" value = "1"
229         Option "Aripirazol" value = "2"
230         Option "Cariprazin" value = "3"
231         Option "Clozapin" value = "4"
232         Option "Haloperidol" value = "5"
233         Option "Olanzapin" value = "6"
234         Option "Paliperidon" value = "7"
```

A Appendices

```
235         Option "Risperidon" value = "8"
236         Option "Perphenazin" value = "9"
237         Option "Quetiapin" value = "10"
238         Option "Sertindol" value = "11"
239         Option "Ziprasidon" value = "12"

241     InputField PsiacCheck
242         #longtext
243         label = 'PSIAC Verification of Conflicted Drugs'

245     InputField AvoidSideEffect
246         #left #multipleChoice #notmandatory
247         Question = 'Side Effects to be Avoided:'
248         Option 'Dry Mouth, Blurred Vision, Constipation' value = '1'
249     ,
250         Option 'Muscular Stiffness, Movement Disorders, Tremor'
value = '2'
251         Option 'Reduction of Sexual Desire, Sexual Dysfunction,
Menstrual Cramps' value = '3'
252         Option 'Weight Gain' value = '4'
253         Option 'Fatigue' value = '5'
254         Option 'Restless Legs' value = '6'
255         Option 'Epilepsy' value = '7'

256     InputField TolerableSideEffect
257         #right #multipleChoice #notmandatory
258         Question = 'Side Effects to be Tolerated:'
259         Option 'Dry Mouth, Blurred Vision, Constipation' value = '1'
260     ,
261         Option 'Muscular Stiffness, Movement Disorders, Tremor'
value = '2'
262         Option 'Reduction of Sexual Desire, Sexual Dysfunction,
Menstrual Cramps' value = '3'
263         Option 'Weight Gain' value = '4'
264         Option 'Fatigue' value = '5'
265         Option 'Restless Legs' value = '6'

266     InputField Comment
267         #notmandatory #stretched
268         label = 'Comment:'

270 Stage Discharge
271     #mandatory #manualActivate
272     owner = 'Setting.CaseOwner'
273     label = 'Discharge'

275     precondition
276         previousStep = 'Identification'

278     HumanTask DischargePatient
279         #mandatory
280         owner = 'Setting.CaseOwner'
281         label = "Discharge Patient"

283     Form DischargeForm
284         InputField DoctorNote
285             #text
286             label = "Post-Treatment Recommendation:"
```

Listing A.3: Acadela code to define the CP of Selection of Antipsychotics for Schizophrenia

A.2.3 Diagnosis of Class II Smoke Inhalation Injury

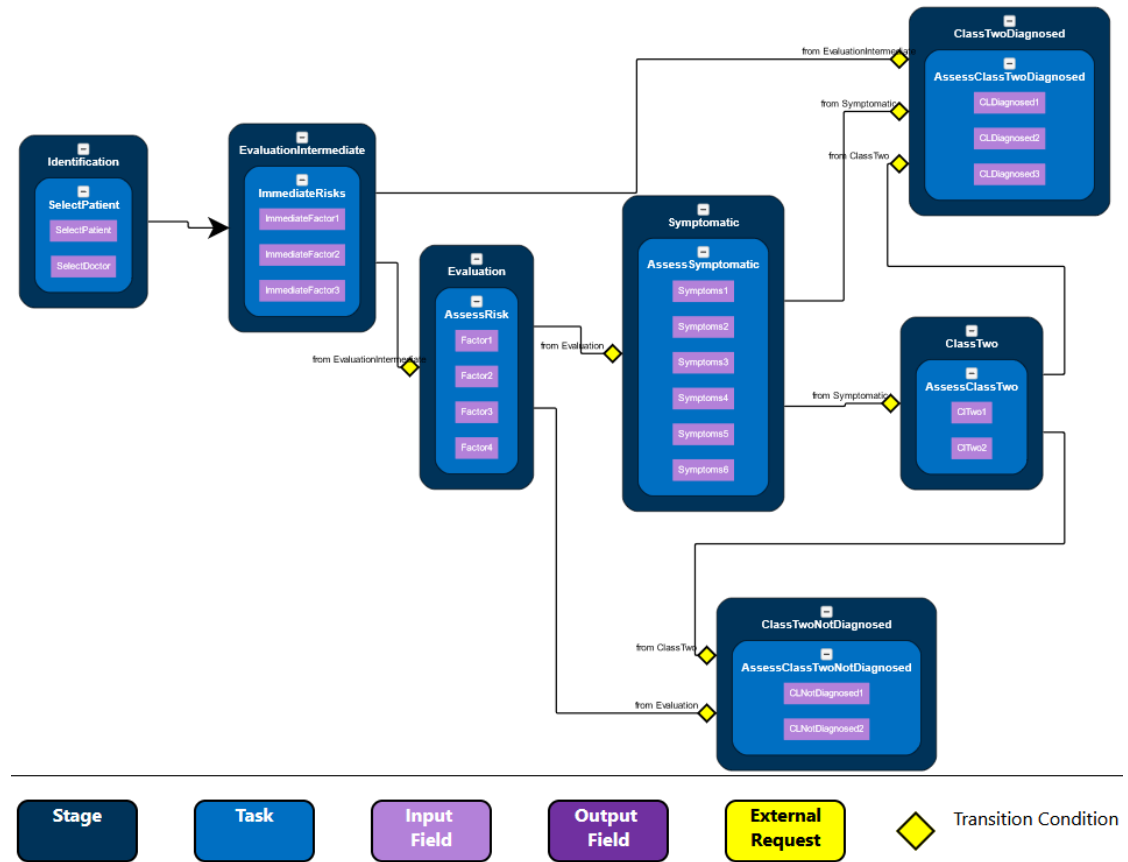


Figure A.3: Visualization of the CP model generated by Listing A.4.

```

1 workspace Demo
2   define case SPI_SmokeInhalation
3     prefix = 'SPI'
4     version = 1
5     label = 'Assessment Of Smoke Inhalation Injury'
6
7     Responsibilities
8       group DemoPhysicians name = 'Demo Physician'
9       group DemoClinicians name = 'Demo Clinician'
10      group DemoProfessionals name = 'Demo Professional'
11      group DemoPatients name = 'Demo Patient'
12      group DemoNurses name = 'Demo Nurse'
13
14     Setting
15     CaseOwner DemoProfessionals #exactlyOne
16       label = 'Demo Professionals'
17     Attribute WorkplanDueDate
18       #exactlyOne #date.#after(TODAY)
19       label = 'Workplan Due Date'
20       externalId = 'dueDateConnie'
21
22     CasePatient DemoPatients #exactlyOne
23       label = 'Patient'
24
25     Attribute Clinician
26       #exactlyOne #Link.#Users(DemoClinicians)

```

A Appendices

```
27     label = 'Clinician'
28
29     Attribute Nurse
30         #exactlyOne #Link.#Users(DemoNurses)
31         label = 'Nurse'
32     SummaryPanel
33         Section Risks #left
34             label = "Immediate Signs for Class II?"
35             InfoPath EvaluationIntermediate.ImmediateRisks.ImmediateFactor1
36             InfoPath EvaluationIntermediate.ImmediateRisks.ImmediateFactor2
37             InfoPath EvaluationIntermediate.ImmediateRisks.ImmediateFactor3
38         Section Risks #left
39             label = "Potential Signs for Class II?"
40             InfoPath Evaluation.AssessRisk.Factor1
41             InfoPath Evaluation.AssessRisk.Factor2
42             InfoPath Evaluation.AssessRisk.Factor3
43             InfoPath Evaluation.AssessRisk.Factor4
44         Section Risks #left
45             label = "Symptoms for Class II?"
46             InfoPath Symptomatic.AssessSymptomatic.Symptoms1
47             InfoPath Symptomatic.AssessSymptomatic.Symptoms2
48             InfoPath Symptomatic.AssessSymptomatic.Symptoms3
49             InfoPath Symptomatic.AssessSymptomatic.Symptoms4
50             InfoPath Symptomatic.AssessSymptomatic.Symptoms5
51             InfoPath Symptomatic.AssessSymptomatic.Symptoms6
52         Section Risks #left
53             label = "Results of laryngoscopy"
54             InfoPath ClassTwo.AssessClassTwo.CLTwo1
55             InfoPath ClassTwo.AssessClassTwo.CLTwo2
56         Section Risks #right
57             label = "Actions taken for Class II diagnosed patient"
58             InfoPath ClassTwoDiagnosed.AssessClassTwoDiagnosed.CLDiagnosed1
59             InfoPath ClassTwoDiagnosed.AssessClassTwoDiagnosed.CLDiagnosed2
60             InfoPath ClassTwoDiagnosed.AssessClassTwoDiagnosed.CLDiagnosed3
61         Section Risks #right
62             label = "Actions taken for Class II not diagnosed patient"
63             InfoPath ClassTwoNotDiagnosed.AssessClassTwoNotDiagnosed.
64             CLNotDiagnosed1
65             InfoPath ClassTwoNotDiagnosed.AssessClassTwoNotDiagnosed.
66             CLNotDiagnosed2
67
68     Stage Identification
69         #mandatory
70         owner = 'Setting.CaseOwner'
71         label = 'Identification'
72
73     HumanTask SelectPatient
74         #mandatory
75         label = 'Assign Patient'
76         owner = 'Setting.Nurse'
77         dueDateRef = 'Setting.WorkplanDueDate'
78         externalId = 'SelectPatient'
79
80     Form PatientAssignForm
81         #mandatory
82
83         InputField SelectPatient
84             #custom
85             ElementPath = "Setting.CasePatient"
86             label = "Assigned Patient"
```

A Appendices

```
86     InputField SelectDoctor
87         #custom
88         ElementPath = "Setting.Clinician"
89         label = "Assigned Clinician"

91     Stage EvaluationIntermediate
92         #mandatory
93         owner = 'Setting.Clinician'
94         label = 'Intermediate Signs'

96     Precondition
97         previousStep = 'Identification'

99     HumanTask ImmediateRisks
100         #mandatory #exactlyOne
101         label = 'Assess immediate signs for smoke inhalation injury (
Class II)'
102         owner = 'Setting.Clinician'
103         dueDateRef = 'Setting.WorkplanDueDate'
104         externalId = 'ImmediateRisks'

106     Form ImmediateRisksForm

108         InputField ImmediateFactor1
109             #singlechoice
110             question = 'Visible burns or edema of the oropharynx'
111             Option 'No' value='0'
112             Option 'Yes' value='1'
113         InputField ImmediateFactor2
114             #singlechoice
115             question = 'Full thickness nasolabial burns'
116             Option 'No' value='0'
117             Option 'Yes' value='1'
118         InputField ImmediateFactor3
119             #singlechoice
120             question = 'Circumferential neck burns'
121             Option 'No' value='0'
122             Option 'Yes' value='1'
123     Stage Evaluation
124         #mandatory
125         owner = 'Setting.Clinician'
126         label = 'Potential Signs'

128     Precondition
129         previousStep = 'EvaluationIntermediate'
130         condition = 'EvaluationIntermediate.ImmediateRisks.
ImmediateFactor1 + EvaluationIntermediate.ImmediateRisks.
ImmediateFactor2 + EvaluationIntermediate.ImmediateRisks.
ImmediateFactor3 = 0'

132     HumanTask AssessRisk
133         #mandatory #exactlyOne
134         label = 'Assess potential signs for smoke inhalation injury (
Class II)'
135         owner = 'Setting.Clinician'
136         dueDateRef = 'Setting.WorkplanDueDate'
137         externalId = 'AssessRisk'

139     Form AssessRiskForm
140         InputField Factor1
141             #singlechoice
```

A Appendices

```
142         question = 'Burns in a closed space'
143         Option 'No' value='0'
144         Option 'Yes' value='1'
145     InputField Factor2
146         #singlechoice
147         question = 'Singed nasal hair'
148         Option 'No' value='0'
149         Option 'Yes' value='1'
150     InputField Factor3
151         #singlechoice
152         question = 'Facial burns'
153         Option 'No' value='0'
154         Option 'Yes' value='1'

156     InputField Factor4
157         #singlechoice
158         question = 'Soot in the mouth'
159         Option 'No' value='0'
160         Option 'Yes' value='1'

162     Stage Symptomatic
163         #mandatory
164         owner = 'Setting.Clinician'
165         label = 'Symtopmatic'
166         Precondition
167             previousStep = 'Evaluation'
168             condition = 'Evaluation.AssessRisk.Factor1 + Evaluation.
AssessRisk.Factor2
169                 + Evaluation.AssessRisk.Factor3 + Evaluation.AssessRisk.
Factor4>0'

171     HumanTask AssessSymptomatic
172         #mandatory #exactlyOne
173         label = 'Evaluate symptoms'
174         owner = 'Setting.Clinician'
175         dueDateRef = 'Setting.WorkplanDueDate'
176         externalId = 'Symptomatic'

178     Form SymptomaticForm
179         InputField Symptoms1
180             #singlechoice
181             question = 'signss of respiratory'
182             Option 'No' value='0'
183             Option 'Yes' value='1'
184         InputField Symptoms2
185             #singlechoice
186             question = 'throat pain'
187             Option 'No' value='0'
188             Option 'Yes' value='1'
189         InputField Symptoms3
190             #singlechoice
191             question = 'odynophagia'
192             Option 'No' value='0'
193             Option 'Yes' value='1'
194         InputField Symptoms4
195             #singlechoice
196             question = 'drooling'
197             Option 'No' value='0'
198             Option 'Yes' value='1'
199         InputField Symptoms5
200             #singlechoice
```

A Appendices

```
201         question = 'stridor'
202         Option 'No' value='0'
203         Option 'Yes' value='1'
204     InputField Symptoms6
205         #singlechoice
206         question = 'hoarseness'
207         Option 'No' value='0'
208         Option 'Yes' value='1'
209 Stage ClassTwo
210     #mandatory
211     owner = 'Setting.Clinician'
212     label = 'Laryngoscopy'
214
215     Precondition
216         previousStep = 'Symptomatic'
217         condition = 'Symptomatic.AssessSymptomatic.Symptoms1 +
218 Symptomatic.AssessSymptomatic.Symptoms2 + Symptomatic.
219 AssessSymptomatic.Symptoms3 + Symptomatic.AssessSymptomatic.
220 Symptoms4 + Symptomatic.AssessSymptomatic.Symptoms5 + Symptomatic.
221 AssessSymptomatic.Symptoms6=0'
223
224     HumanTask AssessClassTwo
225     #mandatory #exactlyOne
226     label = 'Direct or indirect laryngoscopy'
227     owner = 'Setting.Clinician'
228     dueDateRef = 'Setting.WorkplanDueDate'
229     externalId = 'AssessClassTwo'
231
232     Form ClassTwoForm
233     InputField ClTwo1
234         #singlechoice
235         question = 'Erythema at upper airway'
236         Option 'No' value='0'
237         Option 'Yes' value='1'
238     InputField ClTwo2
239         #singlechoice
240         question = 'Blisters of the palate'
241         Option 'No' value='0'
242         Option 'Yes' value='1'
244
245     Stage ClassTwoDiagnosed
246     #mandatory
247     owner = 'Setting.Clinician'
248     label = 'Class II Diagnosed'
250
251     Precondition
252         previousStep = 'EvaluationIntermediate'
253         condition = 'EvaluationIntermediate.ImmediateRisks.
254 ImmediateFactor1 + EvaluationIntermediate.ImmediateRisks.
255 ImmediateFactor2 + EvaluationIntermediate.ImmediateRisks.
256 ImmediateFactor3 > 0'
258     Precondition
259         previousStep = 'Symptomatic'
260         condition = 'Symptomatic.AssessSymptomatic.Symptoms1 +
261 Symptomatic.AssessSymptomatic.Symptoms2 + Symptomatic.
262 AssessSymptomatic.Symptoms3 + Symptomatic.AssessSymptomatic.
263 Symptoms4 + Symptomatic.AssessSymptomatic.Symptoms5 + Symptomatic.
264 AssessSymptomatic.Symptoms6>0'
266     Precondition
267         previousStep = 'ClassTwo'
```


A Appendices

```
250     condition = 'ClassTwo.AssessClassTwo.ClTwo1 + ClassTwo.
AssessClassTwo.ClTwo2>0'

252     HumanTask AssessClassTwoDiagnosed
253     #mandatory #exactlyOne
254     label = 'Next steps for patient Diagnosed with Class II'
255     owner = 'Setting.Clinician'
256     dueDateRef = 'Setting.WorkplanDueDate'
257     externalId = 'AssessClassTwoDiagnosed'

259     Form ClassTwoDiagnosedForm
260     InputField CLDiagnosed1
261     #singlechoice
262     question = 'Early intubation for airway protection'
263     Option 'Yes' value='0'
264     Option 'Yes' value='1'
265     InputField CLDiagnosed2
266     #singlechoice
267     question = 'Admit to ICU for bronchoscopy or consider to
transfer to burn center'
268     Option 'Yes' value='0'
269     Option 'Yes' value='1'
270     InputField CLDiagnosed3
271     #singlechoice
272     question = 'Transfer to burn center'
273     Option 'Yes' value='0'
274     Option 'Yes' value='1'

276     Stage ClassTwoNotDiagnosed
277     #mandatory
278     owner = 'Setting.Clinician'
279     label = 'Class II not Diagnosed'

281     Precondition
282     previousStep = 'Evaluation'
283     condition = 'Evaluation.AssessRisk.Factor1 + Evaluation.
AssessRisk.Factor2 + Evaluation.AssessRisk.Factor3 + Evaluation.
AssessRisk.Factor4 = 0'

284     Precondition
285     previousStep = 'ClassTwo'
286     condition = 'ClassTwo.AssessClassTwo.ClTwo1 + ClassTwo.
AssessClassTwo.ClTwo2 =0'

288     HumanTask AssessClassTwoNotDiagnosed
289     #mandatory #exactlyOne
290     label = 'Next steps for patient not Diagnosed with Class II'
291     owner = 'Setting.Clinician'
292     dueDateRef = 'Setting.WorkplanDueDate'
293     externalId = 'AssessClassTwoNotDiagnosed'

295     Form ClassTwoNotDiagnosedForm
296     InputField CLNotDiagnosed1
297     #singlechoice
298     question = 'If high-risk, consider 24-hour observation to
rule out lower airway injury'
299     Option 'No' value='0'
300     Option 'Yes' value='1'
301     InputField CLNotDiagnosed2
302     #singlechoice
303     question = 'If high-risk, bronchoscopy to rule out lower
airway injury'
```

```
304 Option 'No' value='0'
305 Option 'Yes' value='1'
```

Listing A.4: Acadela code to define the CP of Smoke Inhalation Injury

A.2.4 Cervical Cancer Diagnosis

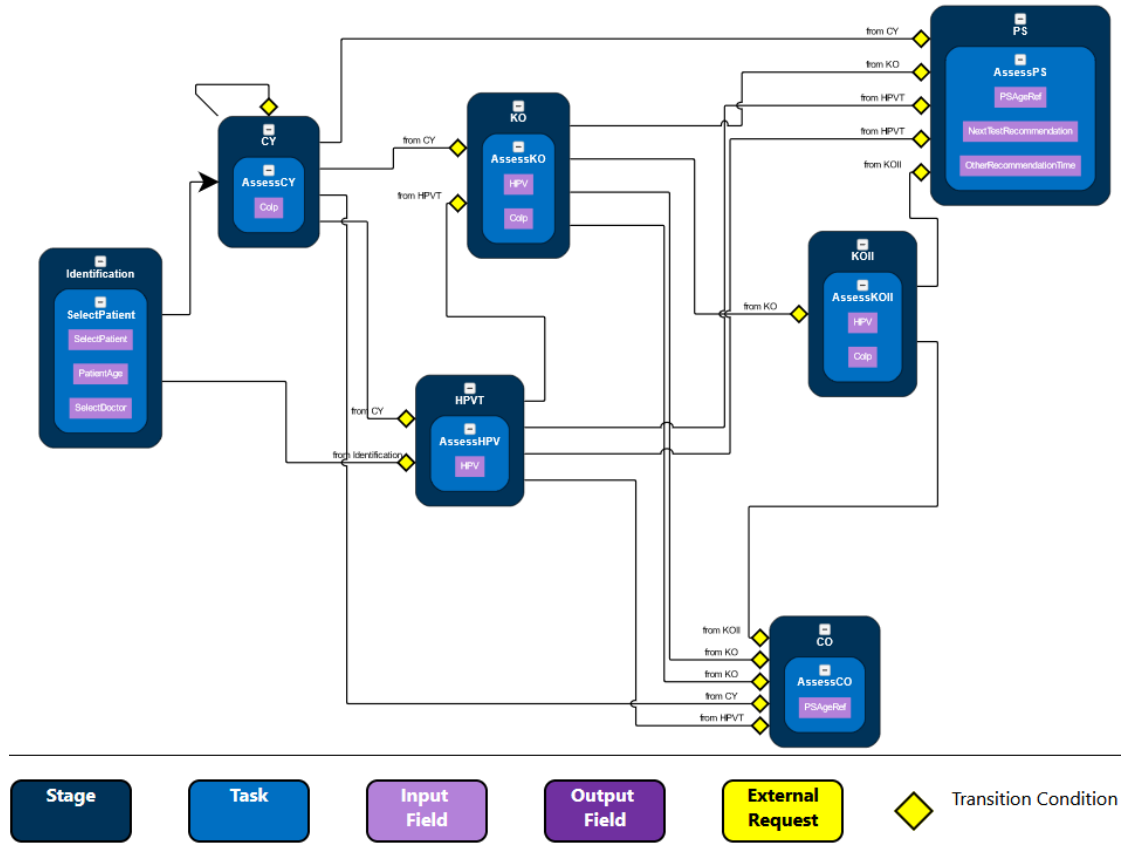


Figure A.4: Visualization of the CP model generated by Listing A.5.

```
1 workspace Demo
2 define case SPI_CervicalCancer
3   prefix = 'SPI'
4   version = 12
5   label = 'Cervical Cancer'
6
7   Responsibilities
8     group DemoPhysicians name = 'Demo Physician'
9     group DemoClinicians name = 'Demo Clinician'
10    group DemoProfessionals name = 'Demo Professional'
11    group DemoPatients name = 'Demo Patient'
12    group DemoNurses name = 'Demo Nurse'
13
14   Setting
15     CaseOwner DemoProfessionals #exactlyOne
16     label = 'Demo Professionals'
17     Attribute WorkplanDueDate
18     #exactlyOne #date.#after(TODAY)
19     label = 'Workplan Due Date'
20     externalId = 'dueDateConnie'
```

A Appendices

```
22     CasePatient DemoPatients #exactlyOne
23         label = 'Patient'

25     Attribute Clinician
26         #exactlyOne #Link.#Users(DemoClinicians)
27         label = 'Clinician'

29     Attribute Nurse
30         #exactlyOne #Link.#Users(DemoNurses)
31         label = 'Nurse'

33     SummaryPanel
34         Section Risks #left
35         label = "Immediate Signs for Class II?"
36         InfoPath Identification.SelectPatient.PatientAge

38     Stage Identification
39         #mandatory
40         owner = 'Setting.CaseOwner'
41         label = 'Identification'

43     HumanTask SelectPatient
44         #mandatory
45         label = 'Assign Patient'
46         owner = 'Setting.Nurse'
47         dueDateRef = 'Setting.WorkplanDueDate'
48         externalId = 'SelectPatient'

50     Form PatientAssignForm
51         #mandatory

53         InputField SelectPatient
54             #custom
55             ElementPath = "Setting.CasePatient"
56             label = "Assigned Patient"

58         InputField PatientAge
59             #number
60             label = "Patient age"

62         InputField SelectDoctor
63             #custom
64             ElementPath = "Setting.Clinician"
65             label = "Assigned Clinician"

67     Stage CY
68         #mandatory #repeatserial
69         owner = 'Setting.Clinician'
70         label = 'Cytological-Testing'

72     Precondition
73         previousStep = 'Identification'

75     Precondition
76         previousStep = 'CY'
77         condition = '(CY.AssessCY.Colp = 2 or CY.AssessCY.Colp = 3) and
78         Identification.SelectPatient.PatientAge < 30'

79     HumanTask AssessCY
80         #mandatory
```

A Appendices

```
81     label = 'Evaluate test results'
82     owner = 'Setting.Clinician'
83     dueDateRef = 'Setting.WorkplanDueDate'
84     externalId = 'AssessRisk'

86     Form CYForm
87         InputField Colp
88             #singlechoice
89                 question = 'Cytological testing'
90                 Option 'Pap I' value='0'
91                 Option 'Pap II-a' value='1'
92                 Option 'Pap II-p,g' value='2'
93                 Option 'Pap IIID-1' value='3'
94                 Option 'Pap IIID-2' value='4'
95                 Option 'Pap III-p,g' value='5'
96                 Option 'Pap IV' value='6'
97                 Option 'Pap V' value='7'
98     Stage HPVT
99         #mandatory
100         owner = 'Setting.Clinician'
101         label = 'HPV-Testing'

103     Precondition
104         previousStep = 'CY'
105         //condition = '(CY.AssessCY.Colp = 2 or CY.AssessCY.Colp = 3)
and Identification.SelectPatient.PatientAge > 29'
106         condition = '(CY.AssessCY.Colp = 2 or CY.AssessCY.Colp = 3) and
(Identification.SelectPatient.PatientAge >= 30 and Identification.
SelectPatient.PatientAge <= 34)'

108     Precondition
109         previousStep = 'Identification'
110         condition = 'Identification.SelectPatient.PatientAge >= 35'

113     HumanTask AssessHPV
114         #mandatory
115         label = 'Evaluate test results'
116         owner = 'Setting.Clinician'
117         dueDateRef = 'Setting.WorkplanDueDate'
118         externalId = 'AssessRisk'

120     Form HPVForm
121         InputField HPV
122             #singlechoice
123                 question = 'Cytological testing'
124                 Option 'Negative' value='0'
125                 Option 'Positive' value='1'

127     Stage KO
128         #mandatory
129         owner = 'Setting.Clinician'
130         label = 'KO-Testing'

132     //Precondition
133     // previousStep = 'Identification'
134     // condition = 'Identification.SelectPatient.PatientAge > 34'

136     Precondition
137         previousStep = 'HPVT'
138         previousStep = 'CY'
```

A Appendices

```

139         condition = 'Identification.SelectPatient.PatientAge >= 35 and
((HPVT.AssessHPV.HPV = 1 and CY.AssessCY.Colp = 0) or (HPVT.
AssessHPV.HPV = 0 and CY.AssessCY.Colp = 3))'

141     HumanTask AssessKO
142         #mandatory #exactlyOne
143         label = 'Evaluate test results'
144         owner = 'Setting.Clinician'
145         dueDateRef = 'Setting.WorkplanDueDate'
146         externalId = 'AssessRisk'

148     Form KOForm
149         InputField HPV
150         #singlechoice
151             question = 'HPV'
152             Option 'Negative' value='0'
153             Option 'Positive' value='1'
154         InputField Colp
155         #singlechoice
156             question = 'Cytological testing'
157             Option 'Pap I' value='0'
158             Option 'Pap II-p,g' value='2'
159             Option 'Pap IIID-1' value='3'
160             Option 'Pap IIID-2' value='4'
161             Option 'Pap III-p,g' value='5'
162             Option 'Pap IV' value='6'
163             Option 'Pap V' value='7'
164     Stage KOII
165         #mandatory
166         owner = 'Setting.Clinician'
167         label = 'KO-Testing II'

169     Precondition
170         previousStep = 'KO'
171         condition = 'KO.AssessKO.HPV = 1 and KO.AssessKO.Colp = 0 or KO
.AssessKO.HPV = 0 and KO.AssessKO.Colp = 3'

173     HumanTask AssessKOII
174         #mandatory #exactlyOne
175         label = 'Evaluate test results'
176         owner = 'Setting.Clinician'
177         dueDateRef = 'Setting.WorkplanDueDate'
178         externalId = 'AssessRisk'

180     Form KOIIForm
181         InputField HPV
182         #singlechoice
183             question = 'HPV'
184             Option 'Negative' value='0'
185             Option 'Positive' value='1'
186         InputField Colp
187         #singlechoice
188             question = 'Cytological testing'
189             Option 'Pap I' value='0'
190             Option 'Pap II-a' value='1'
191             Option 'Pap II-p,g' value='2'
192             Option 'Pap IIID-1' value='3'
193             Option 'Pap IIID-2' value='4'
194             Option 'Pap III-p,g' value='5'
195             Option 'Pap IV' value='6'
196             Option 'Pap V' value='7'

```

A Appendices

```
199 Stage CO
200   #mandatory
201   owner = 'Setting.Clinician'
202   label = 'Colposcopy'

204   Precondition
205     previousStep = 'CY'
206     condition = 'CY.AssessCY.Colp = 5 or CY.AssessCY.Colp = 4 or CY
207 .AssessCY.Colp = 6 or CY.AssessCY.Colp = 7'
207   Precondition
208     previousStep = 'KO'
209     condition = 'KO.AssessKO.HPV = 1 and KO.AssessKO.Colp = 3 or KO
210 .AssessKO.HPV = 1 and KO.AssessKO.Colp = 2'
210   Precondition
211     previousStep = 'KO'
212     condition = 'KO.AssessKO.Colp = 5 or KO.AssessKO.Colp = 4 or KO
213 .AssessKO.Colp = 6 or KO.AssessKO.Colp = 7'
213   Precondition
214     previousStep = 'KOII'
215     condition = 'KOII.AssessKOII.HPV = 1 or KOII.AssessKOII.Colp >
216 2'
216   Precondition
217     previousStep = 'HPVT'
218     condition = 'HPVT.AssessHPV.HPV = 1'

220   HumanTask AssessCO
221     #mandatory #exactlyOne
222     label = 'Recommendation: Colposcopy'
223     owner = 'Setting.Clinician'
224     dueDateRef = 'Setting.WorkplanDueDate'
225     externalId = 'AssessRisk'

227   Form COForm
228     InputField PSAgeRef
229     #custom
230     ElementPath = "Identification.SelectPatient.PatientAge"
231     label = "Patient Age"
232 Stage PS
233   #mandatory
234   owner = 'Setting.Clinician'
235   label = 'Primary Screening'

237   Precondition
238     previousStep = 'HPVT'
239     condition = 'HPVT.AssessHPV.HPV = 0 and (CY.AssessCY.Colp = 0
240 or CY.AssessCY.Colp = 2) and Identification.SelectPatient.
241 PatientAge >= 35'

241   Precondition
242     previousStep = 'HPVT'
243     condition = 'HPVT.AssessHPV.HPV = 0 and (CY.AssessCY.Colp = 2
244 or CY.AssessCY.Colp = 3) and (Identification.SelectPatient.
245 PatientAge >= 30 and Identification.SelectPatient.PatientAge <= 34)
246 ,

245   Precondition
246     previousStep = 'CY'
247     condition = '(CY.AssessCY.Colp = 0 or CY.AssessCY.Colp = 1) and
248 Identification.SelectPatient.PatientAge < 35'
```

A Appendices

```
249     Precondition
250         previousStep = 'KO'
251         condition = 'KO.AssessKO.HPV = 0 and (KO.AssessKO.Colp = 0 or
KO.AssessKO.Colp = 2) and Identification.SelectPatient.PatientAge
>= 35'
252     Precondition
253         previousStep = 'KOII'
254         condition = 'KOII.AssessKOII.HPV = 0 and KOII.AssessKOII.Colp <
3'
256     HumanTask AssessPS
257         #mandatory #exactlyOne
258         label = 'Recommendation: Primary Screening'
259         owner = 'Setting.Clinician'
260         dueDateRef = 'Setting.WorkplanDueDate'
261         externalId = 'AssessRisk'
263     Form PSForm
264         #mandatory
266     InputField PSAgeRef
267         #custom
268         ElementPath = "Identification.SelectPatient.PatientAge"
269         label = "Patient Age"
271     InputField NextTestRecommendation
272         #singlechoice
273         question = 'Recommended Time for the Next Test:'
274         option 'none' value = '0'
275         option 'every 3 month' value = '1'
276         option 'every 6 month' value = '2'
277         option 'every year' value = '3'
278         option 'every 3 year' value = '4'
279         option 'other' value = '5'
281     InputField OtherRecommendationTime
282         #text
283         label = 'Other Recommendation Time:'
```

Listing A.5: Acadela code to define the CP of Cervical Cancer

A.2.5 Chronic Headache Treatment

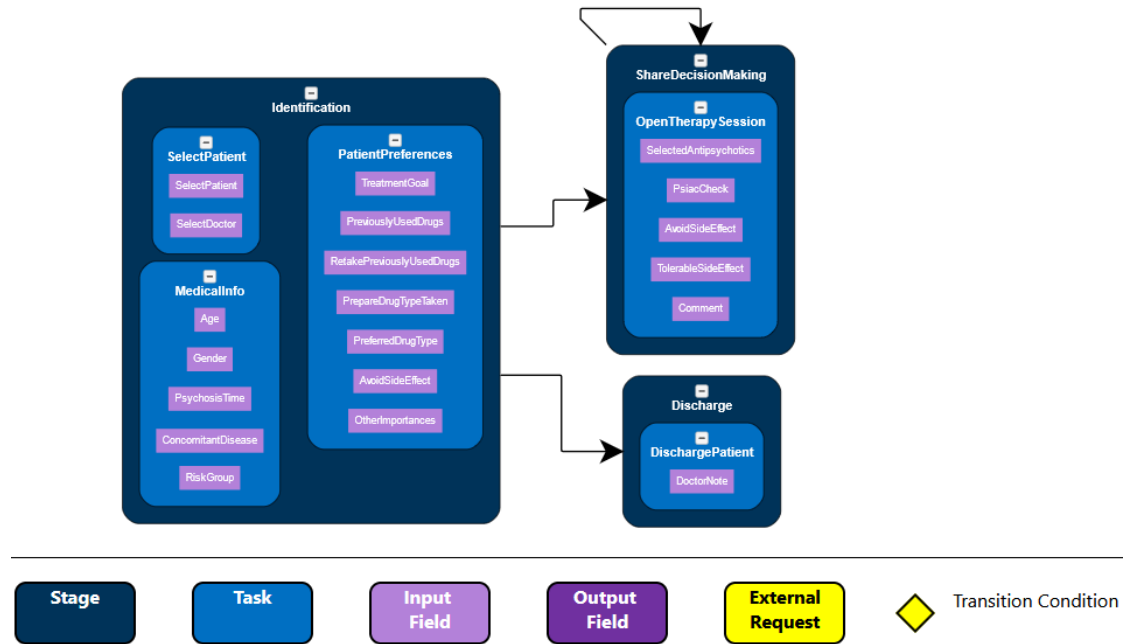


Figure A.5: Visualization of the CP model generated by Listing A.6.

```

1  aca0.1
2  import extfile.redGreenUiRef as rgu
3  import extfile.template.body3ViewsTemplate as bTemplate
4  workspace Demo

6  define case MI1_Headache
7  prefix = 'MI1'
8  version = 5
9  label = 'Chronic Headache Treatment'

11 Responsibilities
12 group DemoPhysicians name = 'Demo Physician' //staticId = 'asdf234'
13 group DemoClinicians name = 'Demo Clinician'
14 group DemoProfessionals name = 'Demo Professional'
15 group DemoPatients name = 'Demo Patient'
16 group DemoNurses name = 'Demo Nurse'

18 Setting
19 CaseOwner DemoClinicians #exactlyOne
20 label = 'MI Clinician'

22 Attribute WorkplanDueDate
23 #exactlyOne #date.#after(TODAY)
24 label = 'Workplan Due Date'

26 CasePatient DemoPatients #exactlyOne
27 label = 'Patient'

29 Attribute Clinician
30 #exactlyOne #Link.#Users(DemoClinicians)
31 label = 'Clinician'

33 SummaryPanel
    
```


A Appendices

```
34 Section PainAreaSummary #left
35     label = "Pain Area:"
36     InfoPath Questioning.QuestionPatientCondition.PainArea

38 Stage Observation
39     #mandatory
40     owner = 'Setting.CaseOwner'
41     label = 'Observation'

43 HumanTask CheckBreath
44     #mandatory
45     label = 'Observe Breath Pattern'
46     dueDateRef = 'Setting.WorkplanDueDate'

48 Form BreathCheckForm
49     #mandatory

51     InputField BreathPattern
52         #singleChoice
53         question = "What is the breathing pattern of the patient?"
54         option 'Light' value = '0'
55         option 'Short' value = '1'
56         option 'Chest breathing' value = '2'
57         option 'Diaphragmatic breathing' value = '3'
58         option 'Deep' value = '4'

60     InputField BreathPatternNote
61         #text
62         question = "What is the breathing pattern of the patient?"
63         option 'Light' value = '0'
64         option 'Short' value = '1'
65         option 'Chest breathing' value = '2'
66         option 'Diaphragmatic breathing' value = '3'
67         option 'Deep' value = '4'

69 HumanTask CheckBehavior
70     #mandatory
71     label = 'Observe Behaviors'
72     dueDateRef = 'Setting.WorkplanDueDate'

74 Form BehaviorCheckForm
75     #mandatory

77     InputField WalkPattern
78         #singleChoice
79         question = "How does the patient walk?"
80         option 'Stiff' value = '0'
81         option 'Fragmentated' value = '1'
82         option 'Bent over' value = '2'
83         option 'Smooth' value = '3'
84         option 'Stretched' value = '4'
85         option 'Heavy' value = '5'
86         option 'Upstraight & head up' value = '6'

88     InputField ShakeHandPattern
89         #multipleChoice #notMandatory
90         question = "How does the patient shake hand?"
91         option 'Weak' value = '0'
92         option 'Normal' value = '1'
93         option 'Strong' value = '2'
94         option 'With eye contact' value = '3'
```

A Appendices

```
95     option 'No eye contact' value = '4',
96     option 'Brief' value = '5'

98     InputField PatientTension
99         #singleChoice
100         question = "How tense is the patient?"
101         option 'Confused' value = '0',
102         option 'Fluffy' value = '1',
103         option 'Relaxed' value = '2',
104         option 'Tense' value = '3',
105         option 'Chaotic' value = '4'

108     Stage Questioning
109         #mandatory
110         owner = 'Setting.Clinician',
111         label = 'Questioning'

113     Precondition
114         previousStep = 'Observation'

116     HumanTask QuestionPatientCondition
117         #mandatory
118         owner = 'Setting.Clinician',
119         dueDateRef = 'Setting.WorkplanDueDate',
120         label = 'Question Patient Condition'

122     Form PatientConditionForm
123         #mandatory
124         InputField HowHeadacheStart
125             #text #left
126             label = "How did it start?"

128         InputField WhenHeadacheStart
129             #text #center
130             label = "When did it start?"

132         InputField HeadacheFrequency
133             #text #left
134             label = "How often is the headache?"

136         InputField PainQuality
137             #multipleChoice #left
138             question = "What is the pain quality?"
139             option "Wavy" value = '1',
140             option "Tingling" value = '2',
141             option "Burning" value = '3',
142             option "Stinging (like a needle pointing)" value = '4',
143             option "Pain around the head" value = '5',
144             option "Pain on the eyes" value = '6'

146         InputField PainArea
147             #multipleChoice #left
148             question = "Where is/are the location of the pain(s)?"
149             option "Shoulder" value = 'SHOULDER',
150             option "In the head" value = 'INHEAD',
151             option "On top of the head" value = 'TOPHEAD',
152             option "Temple" value = 'TEMPLE',
153             option "Forehead" value = 'FOREHEAD',
154             option "Head Crown" value = 'HEADCROWN',
155             option "Nape" value = 'NAPE'
```

A Appendices

```
157     InputField WorseningSituation
158         #multipleChoice
159         question = "What emotion(s) makes your pain worse?"
160         option "Angry" value = '1'
161         option "Fear" value = '2'
162         option "Sad/Sorrow" value = '3'
163         option "Stressful" value = '4'

165     InputField OtherWorseningSituation
166         #text #notmandatory
167         label = "Are their other emotions or things that worsen
your headache?"

169     InputField SleepCondition
170         #singleChoice
171         question = "How is your sleep condition?"
172         option "I got nightmare frequently" value = '1'
173         option "I cannot sleep well" value = '2'
174         option "Rarely sleep well" value = '3'
175         option "Often sleep well" value = '4'
176         option "Good" value = '5'
177         option "Very good" value = '6'

179     InputField WhatMakesBetter
180         #text
181         label = "What can help you feel less painful?"

183     InputField TreatmentProposal
184         #text
185         label = "Proposed Treatment:"

187     OutputField bodystyle
188         #string
189         label = "Message Style"
190         uiRef = "hidden"
191         expression = 'let messageSites = PainArea in
192             let styleShoulder = if messageSites.contains("SHOULDER")
193                 then ".shoulder{fill:orange} "
194                 else "" in

196             let styleTopHead = styleShoulder + if messageSites.
contains("TOPHEAD")
197                 then ".topHead{fill:orange} "
198                 else "" in

200             let styleInHead = styleTopHead + if messageSites.contains
("INHEAD")
201                 then ".inHead{fill:orange} "
202                 else "" in

204             let styleTemple = styleInHead + if messageSites.contains
("TEMPLE")
205                 then ".temple{fill:orange}"
206                 else "" in

208             let styleForehead = styleTemple + if messageSites.
contains("FOREHEAD")
209                 then ".forehead{fill:orange}"
210                 else "" in
```

A Appendices

```
212         let styleHeadCrown = styleForehead + if messageSites.  
contains("HEADCROWN")  
213             then ".headCrown{fill:orange}"  
214             else "" in  
  
216         let styleNape = styleHeadCrown + if messageSites.contains  
("NAPE")  
217             then ".nape{fill:orange}"  
218             else "" in styleNape  
219     ,  
  
221     InputField bodytemplate  
222         #string #exactlyOne  
223         label = "Body Template"  
224         uiRef = 'hidden'  
225         defaultValue = use bTemplate.body3ViewsTemplate  
  
227     OutputField bodyVisual  
228         #string  
229         label = "Potential Massage Points"  
230         uiRef = 'svg'  
231         expression = 'replace(bodytemplate, "dynamicstylevars{}",  
bodystyle)'  
  
233 Stage TreatmentApproval  
234     #mandatory  
235     label = "Treatment Approval"  
  
237     Precondition  
238         previousStep = "Questioning"  
  
240     HumanTask DiscussTreatment  
241     #mandatory  
242     label = "Discuss Treatment With Patient"  
  
244     Form TreatmentDiscussionForm  
245     #mandatory  
246     InputField PatientConsent  
247         #singleChoice  
248         question = "Does the patient agree with the proposed  
treatment?"  
249         option "No" value = '0'  
250         option "Yes" value = '1'  
251         option "Undecided" value = '2'  
252         option "Other" value = '3'  
  
254     InputField DisagreementReason  
255         #text #notmandatory  
256         label = "What is the patient's concern with the treatment  
process?"  
  
258     InputField MessageConsent  
259         #singleChoice  
260         question = "Does the patient agree to massage?"  
261         option "No" value = '0'  
262         option "Yes" value = '1'  
  
264     InputField MassageTime  
265         #text  
266         label = "How much time you would like to massage?"
```

A Appendices

```
268     InputField AcupunctureConsent
269         #singleChoice
270         question = "Does the patient agree with using Acupuncture?"
271         option "No" value = '0'
272         option "Yes" value = '1'

274     InputField AcupunctureTime
275         #text
276         label = "How much time you would like to acupuncture?"

278     InputField GuashaConsent
279         #singleChoice
280         question = "Does the patient agree with using Guasha?"
281         option "No" value = '0'
282         option "Yes" value = '1'

286     Stage Messaging
287         #mandatory
288         label = "Massaging"

290     Precondition
291         previousStep = "TreatmentApproval"
292         condition = "TreatmentApproval.DiscussTreatment.MessageConsent =
1"

294     HumanTask MessageHead
295         #mandatory
296         label = 'Message Head'

298     Form HeadMessageForm
299         #mandatory

301     InputField HeadMessagePosition
302         #multipleChoice
303         question = "Message the following positions:"
304         option "Shoulder" value = 'SHOULDER'
305         option "Center Upper Back" value = 'CENTERUPPERBACK'
306         option "On top of the head" value = 'TOPHEAD'
307         option "Nape" value = 'NAPE'
308         option "Neck" value = 'NECK'
309         option "Jaw" value = 'JAW'
310         option "Upper Eyes" value = 'UPPEREYES'
311         option "Occipital Bone" value = 'OCCIBONE'
312         option "Head Crown" value = "HEADCROWN"
313         option "Close to Kidney" value = "KIDNEY"
314         option "Temple" value = "TEMPLE"
315         option "Forehead" value = "FOREHEAD"
316         option "Others" value = "OTHER"

318     OutputField messageLocationStyle
319         #string
320         label = "Message Style"
321         uiRef = "hidden"
322         expression = 'let messageSites = HeadMessagePosition in
323             let styleShoulder = if messageSites.contains("SHOULDER")
324                 then ".shoulder{fill:orange} "
325                 else "" in
```

A Appendices

```
327         let styleCenterUpperBack = styleShoulder + if
messageSites.contains("CENTERUPPERBACK")
328             then ".centerUpperBack{fill:orange} "
329             else "" in

331         let styleTopHead = styleCenterUpperBack + if messageSites
.contains("TOPHEAD")
332             then ".topHead{fill:orange} "
333             else "" in

335         let styleNeck = styleTopHead + if messageSites.contains("
NECK")
336             then ".neck{fill:orange} "
337             else "" in

339         let styleJaw = styleNeck + if messageSites.contains("JAW
")
340             then ".jaw{fill:orange} "
341             else "" in

343         let styleUpperEyes = styleJaw + if messageSites.contains
("UPPEREYES")
344             then ".upperEyes{fill:orange} "
345             else "" in

347         let styleOcciBone = styleUpperEyes + if messageSites.
contains("OCCIBONE")
348             then ".occipitalBone{fill:orange} "
349             else "" in

351         let styleHeadCrown = styleOcciBone + if messageSites.
contains("HEADCROWN")
352             then ".headCrown{fill:orange}"
353             else "" in

355         let styleKidney = styleHeadCrown + if messageSites.
contains("KIDNEY")
356             then ".kidney{fill:orange}"
357             else "" in

359         let styleTemple = styleKidney + if messageSites.contains
("TEMPLE")
360             then ".temple{fill:orange}"
361             else "" in

363         let styleForehead = styleTemple + if messageSites.
contains("FOREHEAD")
364             then ".forehead{fill:orange}"
365             else "" in

367         let styleNape = styleForehead + if messageSites.contains
("NAPE")
368             then ".nape{fill:orange}"
369             else "" in styleNape
370     ,

372     InputField messageLocationTemplate
373         #string #exactlyOne
374         label = "Meridian Template"
375         uiRef = 'hidden'
376         defaultValue = use bTemplate.body3ViewsTemplate
```

A Appendices

```
378     OutputField messageLocationVisual
379         #string
380         label = "Potential Massage Points"
381         uiRef = 'svg'
382         externalId = 'messageLoc'
383         expression = 'replace(messageLocationTemplate, "
dynamicstylevars{}", messageLocationStyle)''

385     InputField OtherMessagePosition
386         #text #notMandatory
387         label = 'Apply message to the other following positions:''

389     InputField ApplyHeat
390         #singleChoice
391         question = "Using Heat Lamp during the massage:"
392         option "No" value = '0'
393         option "Yes" value = '1'

395 Stage Acupuncture
396     #mandatory
397     label = "Acupuncture"

399     Precondition
400         previousStep = "TreatmentApproval"
401         condition = "TreatmentApproval.DiscussTreatment.
AcupunctureConsent = 1"

403     HumanTask AcupuncturePosition
404         #mandatory
405         label = 'Apply Acupuncture:''

407     Form AcuPosForm
408         #mandatory
409         InputField AcupuncturePos
410         #multiplechoice
411         Question = 'Apply Acupuncture to Positions:''
412         Option "Below Left Ear" value = '1'
413         Option "Below Right Ear" value = '2'
414         Option "Left Nape" value = '3'
415         Option "Right Nape" value = '4'

417 Stage Guasha
418     #mandatory
419     label = "Guasha"

421     Precondition
422         previousStep = "TreatmentApproval"
423         condition = "TreatmentApproval.DiscussTreatment.GuashaConsent = 1
"

425     HumanTask ApplyGuasha
426         #mandatory
427         label = 'Apply Guasha:''

429     Form ApplyGuashaForm
430         #mandatory
431         InputField GuashaUsage
432         #text
433         label = 'Apply Guasha to the following positions:''
```

```

435 Stage Discharge
436     #mandatory
437     label = "Discharge"
438
439     Precondition
440         previousStep = "Observation"
441
442     HumanTask DischargePatient
443         #mandatory
444         label = 'Discharge'
445
446     Form DischargeForm
447         #mandatory
448         InputField DoctorNote
449         #text
450         label = 'Doctor Note:'

```

Listing A.6: Acadela code to define the CP of Chronic Headache

A.3 Code Snippets

A.3.1 Responsibilities Declaration and Assignment

This section shows how to declare and assign Group and User to a phase or activity in Acadela. In Listing A.7 from line 1 to line 4, Acadela declares a Group with ID is *StPaulPhysicians*. The *name* attribute specifies the group name in the database of the e-Health system. We did not assign the group *name* to the group *ID* because the group *name* can contain space, which can cause confusion when referencing the group and complicate the grammar. *staticId* is an optional attribute representing the group ID **in the SACM system**. Similarly, line 5 declares a User group with ID as the username. Lines 7 to 10 set the Case Owner as the *StPaulPhysicians* group declared in line 2. The *Case Owner* is the entity that has both the read and write privileges to data in a case.

```

1 Responsibilities
2   Group StPaulPhysicians
3     name = 'StPaul Physicians'
4     staticId = 'x5saefs'
5   User williamst staticId = '
6     ag25smb'
7
8   Setting
9     CaseOwner StPaulPhysicians
10    #exactlyOne
11    label = 'Physician'

```

Listing A.7: Group and User Declaration Syntax

```

1 Stage Diagnosis
2   owner = 'Setting.
3     CaseOwner'
4   ...

```

Listing A.8: Responsibility Assignment example: appointing the *CaseOwner* in the Setting of Listing A.7 to execute the *Diagnosis* Stage

A.3.2 Dynamic Template Definition

Listing A.9 shows how Acadela defines an InputField to collect message positions (lines 1-6). In the Template definition (lines 8-20), modelers specify 1) CSS style placeholder, in this case is *#dynamicstylevars*; 2) the SVG image data with 3) hidden circles at the corresponding body or head positions in the image using the *cx* and *cy* attributes. The OutputField *messageLocationStyle* defines a dynamic rendering by adding a CSS style to fill the orange color to a circle with a class name matching the selected body part.

A Appendices

The final `OutputField` replaces the CSS style placeholder with the dynamic rendering mechanism defined in the `messageLocationStyle` `OutputField`.

```
1  InputField HeadMessagePosition #multipleChoice
2  question = "Message the following positions:"
3  option "Temple" value = "TEMPLE"
4  option "Nape" value = 'NAPE'
5  option "Jaw" value = 'JAW'
6  // ... - Options for other head or body areas
7  option "Forehead" value = "FOREHEAD"
8
9  InputField messageLocationTemplate #string #exactlyOne // Import SVG
   Template
10 label = "Meridian Template"
11 uiRef = 'hidden'
12 defaultValue = '<svg ...> // SVG metadata attributes
13 <style> #dynamicstylevars{} </style> // style placeholder
14 <g ...> ... </g> // SVG image data
15
16 <circle class="temple" cx="114" cy="57" r="2" fill="none"/>
17 <circle class="temple" cx="144" cy="57" r="2" fill="none"/>
18 <circle class="temple" cx="277" cy="57" r="2" fill="none"/>
19
20 <circle class="nape" cx="431" cy="93" r="4" fill="none"/>
21 <circle class="nape" cx="445" cy="93" r="4" fill="none"/>
22
23 // ... - Other custom shapes and location definition'
24
25 OutputField messageLocationStyle #string // Dynamic Style based on
   inputs
26 label = "Message Style"
27 uiRef = "hidden"
28 expression = 'let messageSites = HeadMessagePosition in
29 let styleTemple = if messageSites.contains("TEMPLE")
30 then ".temple{fill:orange}" else "" in
31 let styleNape = styleTemple + if messageSites.contains("NAPE")
32 then ".nape{fill:orange}" else "" in
33 // ... - code to define style for each body part
34 let styleForehead = styleNeck + if messageSites.contains("FOREHEAD")
35 then ".forehead{fill:orange}"
36 else "" in styleForehead'
37
38 OutputField messageLocationVisual #string
39 label = "Potential Message Points" uiRef = 'svg'
40 // Replace the placeholder in SVG with the dynamic style
41 expression = 'replace(messageLocationTemplate, "#dynamicstylevars{}",
   messageLocationStyle)'
```

Listing A.9: Dynamic Template Rendering Definition code in Acadela

A.4 System Usability Scale Questionnaire

ID	Statement
1	I think that I would like to use this language frequently.
2	I found this language unnecessarily complex.
3	I think that this language is easy to use.
4	I think that I would need assistance to be able to use this language.
5	I found the various functions such as defining elements, importing modules, error validations in this language were well integrated.
6	I thought there was too much inconsistency in this language.
7	I would imagine that most people would learn to use this language very quickly.
8	I found this language very cumbersome to use.
9	I felt very confident using this language.
10	I needed to learn a lot of things before I could get going with this language.

Table A.1: SUS Statements to Rate Acadela Syntax and Error Validator.

A.5 Syntax and Semantic Error Analyzer

A.5.1 Mapping of Violated Rules and their Human-readable Representation

```

1  {
2    "ID": "identifier (ID)",
3    "STRING": 'Text with quotation marks ("", \'\'')',
4    "Eq": "Equal sign (=)",
5    "INT": "Integer (Number)",
6    "STRICTFLOAT": "Number including fraction",
7    "FLOAT": "Number including fraction",
8    "HASH": "Hash sign (#)",
9    "Hash": "Hash sign (#)",
10   "NUMBER": 'Number',
11   "condition": 'condition',
12   '(if)\s': "if",
13   '(else\sif)\s': "else if",
14   '(else)\s': "else",
15   '(and)\s': "and",
16   '(or)\s': "or",
17   "WorkspaceTerm": "Workspace",
18   'CaseTerm': "Case",
19   'SettingTerm': "Setting",
20   'StageTerm': "Stage",
21   'TaskTerm': "Task",
22   "HumanTaskTerm": "HumanTask",
23   "AutoTaskTerm": "AutoTask",
24   "DualTaskTerm": "DualTask",
25   "FormTerm": "Form",
26   "InputFieldTerm": "InputField",
27   "OutputFieldTerm": "OutputField",
28   "HookTerm": "Hook",
29   "UserTerm": "User",

```

A Appendices

```
30 "GroupTerm ":" Group ",
31 "PreconditionTerm ":" Precondition ",
32 "FormTerm ":" Form ",
33 "AttributeTerm ":" Attribute ",
34 }
```

Listing A.10: Key-value pairs of violated rule name and their Translations

A.6 HttpHook Example Content

```
1 {
2   "isOverdue": false ,
3   "stateTransitions": {
4     "ENABLED": {
5       "by": {
6         "id": "2c9480845bee03e7015bfcad28990010",
7         "email": "practitioner.email@clinic.de",
8         "name": "Doctor James",
9         "resourceType": "users"
10      },
11      "date": "2022-11-04 19:00:56.0"
12    },
13    "TERMINATED": {
14      "by": null ,
15      "date": null
16    },
17    "COMPLETED": {
18      "by": {
19        "id": "2c9480845bee03e7015bfcad28990010",
20        "email": "practitioner.email@clinic.de",
21        "name": "Doctor James",
22        "resourceType": "users"
23      },
24      "date": "2022-11-04 19:01:25.0"
25    },
26    "ACTIVE": {
27      "by": {
28        "id": "2c9480845bee03e7015bfcad28990010",
29        "email": "practitioner.email@clinic.de",
30        "name": "Doctor James",
31        "resourceType": "users"
32      },
33      "date": "2022-11-04 19:00:56.0"
34    },
35    "AVAILABLE": {
36      "by": {
37        "id": "2c9480845bee03e7015bfcad28990010",
38        "email": "practitioner.email@clinic.de",
39        "name": "Doctor James",
40        "resourceType": "users"
41      },
42      "date": "2022-11-04 19:00:56.0"
43    }
44  },
45  "parentStage": "1xfdrfg925r16",
46  "client": null ,
47  "workspace": "2c9480885d1737ef015d74deed260006",
48  "next": null ,
```

A Appendices

```
49   "state": "COMPLETED",
50   "iExternalId": null,
51   "externalId": "measureBmiOperation",
52   "isHighlighted": false,
53   "id": "uja8x6ouh6ci",
54   "possibleActions": [
55     "CORRECT"
56   ],
57   "isManualActivation": false,
58   "nrAlerts": 0,
59   "alerts": [],
60   "description": "Measure BMI",
61   "name": "ST1_MeasureBmi",
62   "nrLogs": 0,
63   "dueDate": null,
64   "ownerConstraint": [],
65   "index": 0,
66   "resourceType": "humantasks",
67   "prev": null,
68   "processDefinition": "1b97eyv0qo9ot",
69   "isVisibleOnDashboard": true,
70   "taskParams": [
71     {
72       "position": "STRETCHED",
73       "isDerived": false,
74       "task": "uja8x6ouh6ci",
75       "additionalDescription": null,
76       "multiplicity": "exactlyOne",
77       "attributeType": "number",
78       "externalId": null,
79       "resourceType": "taskparams",
80       "id": "vkdgza9trevj",
81       "attributeTypeConstraints": {},
82       "values": [
83         1.8
84       ],
85       "defaultValues": [],
86       "description": "Height (m):",
87       "isMandatory": true,
88       "name": "Height",
89       "isReadOnly": false,
90       "uiReference": null
91     },
92     {
93       "position": "STRETCHED",
94       "isDerived": false,
95       "task": "uja8x6ouh6ci",
96       "additionalDescription": null,
97       "multiplicity": "exactlyOne",
98       "attributeType": "number",
99       "externalId": null,
100      "resourceType": "taskparams",
101      "id": "1nybzswudgfv2",
102      "attributeTypeConstraints": {},
103      "values": [
104        83
105      ],
106      "defaultValues": [],
107      "description": "Weight (kg):",
108      "isMandatory": true,
109      "name": "Weight",
```

A Appendices

```
110     "isReadOnly": false ,
111     "uiReference": null
112   },
113   {
114     "position": "STRETCHED",
115     "isDerived": true ,
116     "task": "uja8x6ouh6ci",
117     "evaluationError": null ,
118     "additionalDescription": null ,
119     "attributeType": "number",
120     "externalId": " BmiValueExternal",
121     "resourceType": "taskparams",
122     "id": "s11p2i7w4jx3",
123     "values": [
124       26
125     ],
126     "description": "BMI Calculation:",
127     "isMandatory": true ,
128     "name": "BmiScore",
129     "isReadOnly": true ,
130     "uiReference": null
131   }
132 ],
133 "footnote": null ,
134 "repeatable": "ONCE",
135 "isMandatory": true ,
136 "owner": null ,
137 "nrAlertsUnseen": 0,
138 "case": "1oteh56xcg5s1",
139 "isAutoDraft": true ,
140 "mayEdit": true
141 }
```

A.7 Complete SACM Meta-model

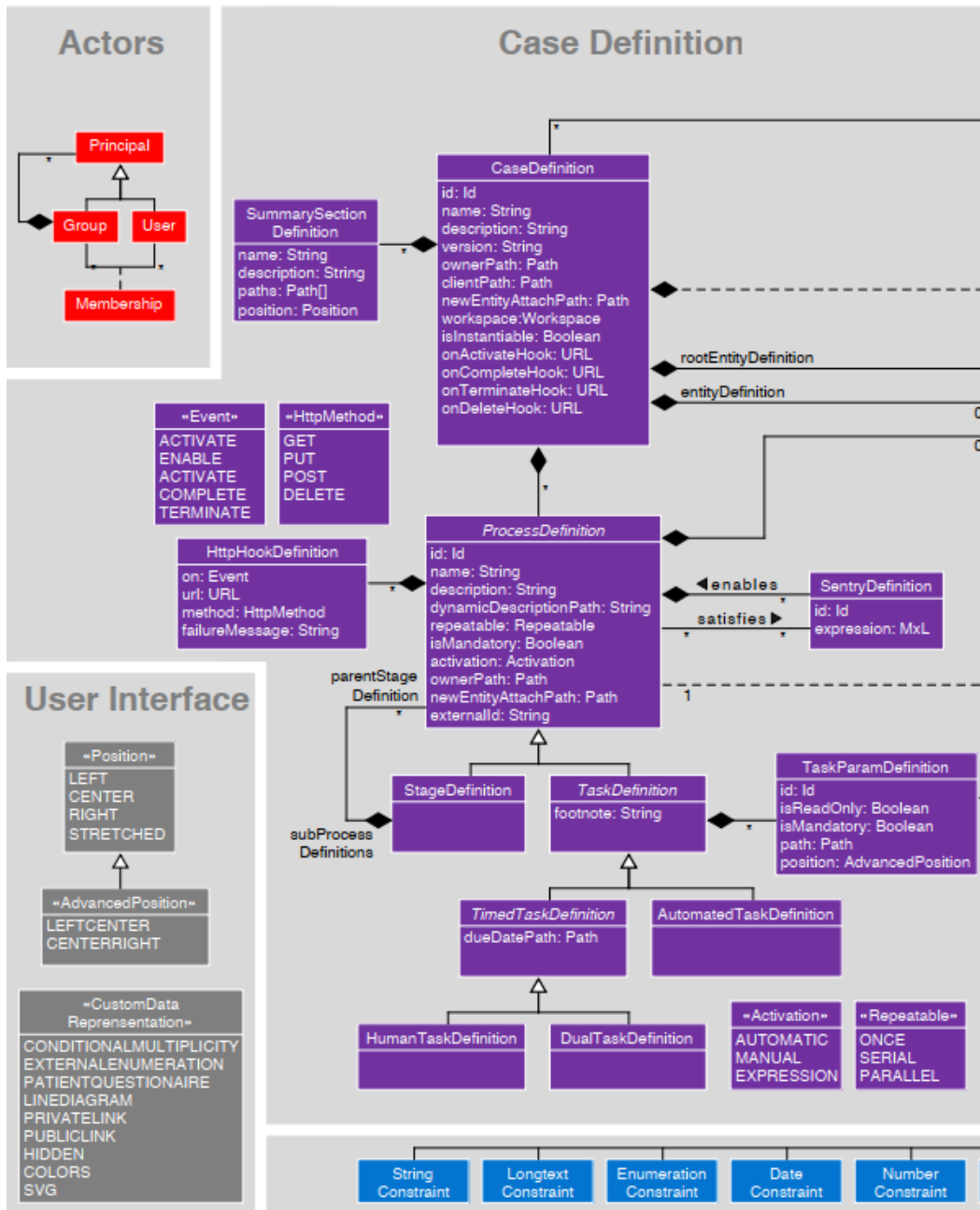


Figure A.6: Detailed meta-model with focus on the case definition (Michel, 2020, p.190).

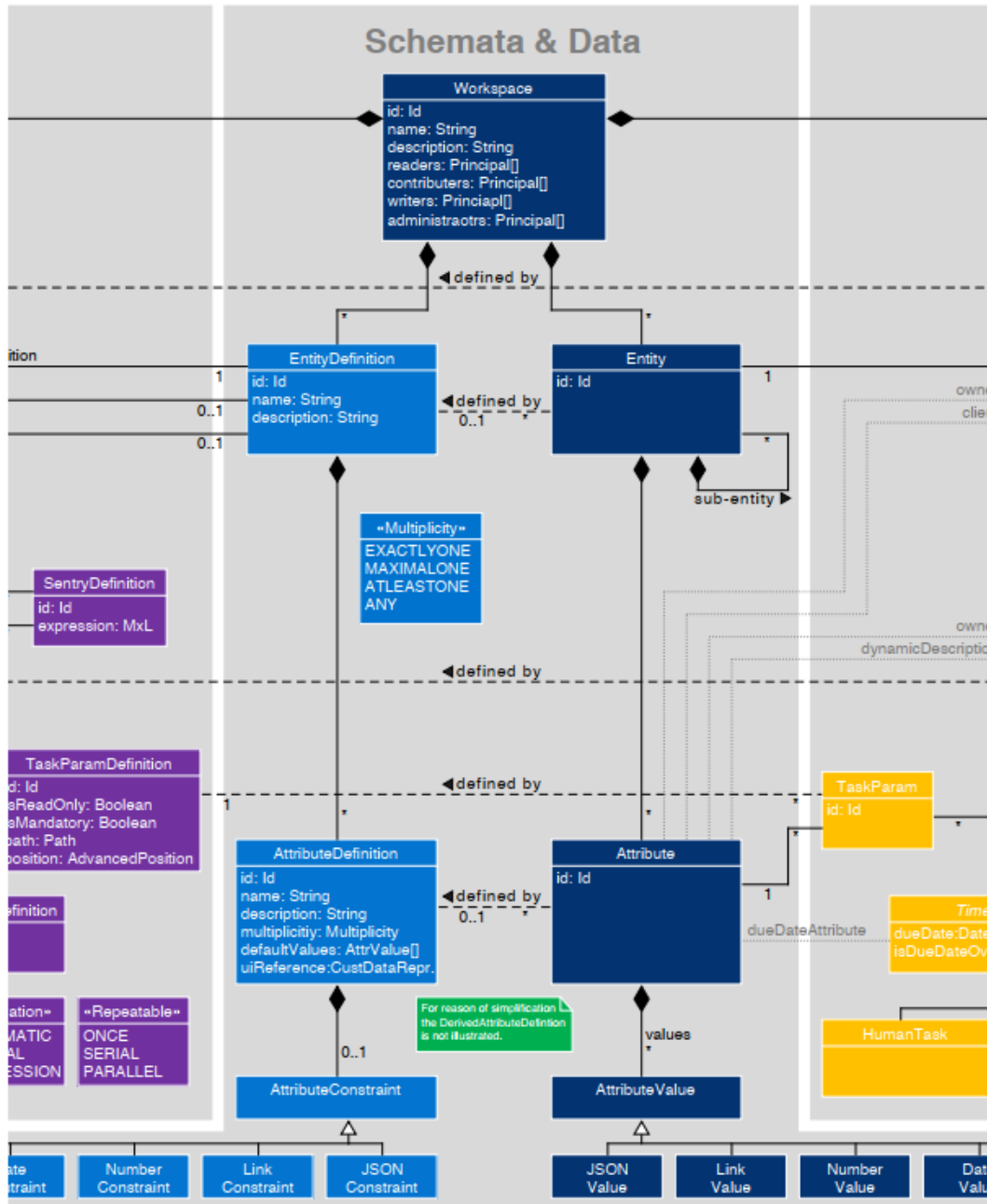


Figure A.7: Detailed meta-model with focus on the schemata and data (Michel, 2020, p.191).

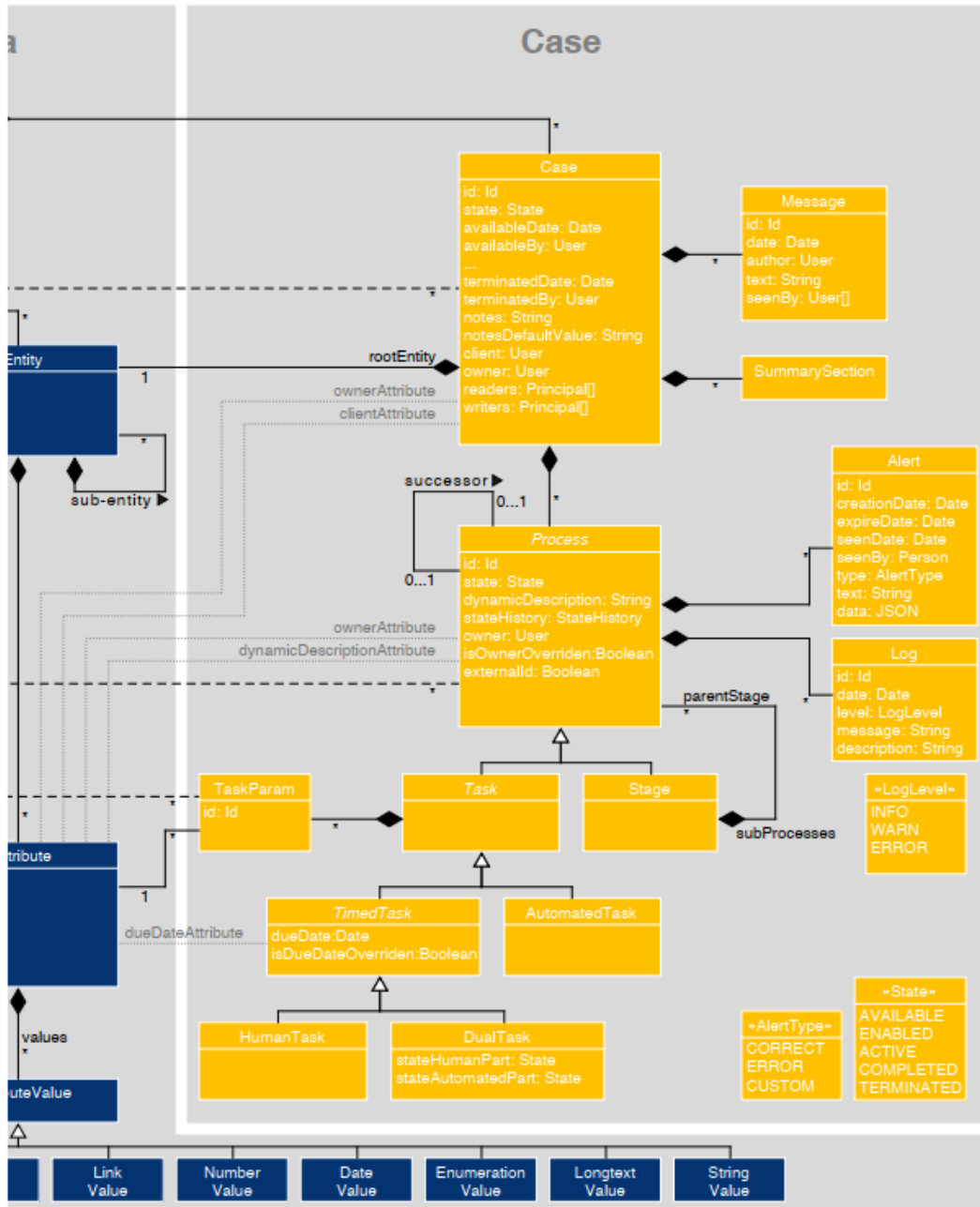


Figure A.8: Detailed meta-model with focus on the case. (Michel, 2020, p.192).