TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik XIX

A Collaborative Purely Meta-Model-Based Adaptive Case Management Approach for Integrated Care

Felix Michel

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Stephan Jonas
Prüfer der Dissertation:	1.	Prof. Dr. Florian Matthes
	2.	Prof. Dr. Martin Bichler

Die Dissertation wurde am 11.12.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 02.06.2020 angenommen.

Zusammenfassung

Heutzutage wird die integrierte Versorgung als vielversprechender Ansatz für patientenorientierte Behandlungen anerkannt, während eine angemessene Softwareunterstützung für die integrierte Versorgung noch fehlt. Herausforderungen sind i) kontextabhängige, unvorhersehbare Behandlungen, ii) semantischer Informationsaustausch bzw. Systeminteroperabilität und iii) Koordination zwischen mehreren Organisationen und verschiedenen Rollen. Papierbasierte Ansätze verhindern eine ausreichende Zusammenarbeit zwischen Pflegefachleuten, während traditionelle Implementierungen, die eine Workflow-Engine mit einem fest verdrahteten Frontend kombinieren, unzureichend sind angesichts des Aufwands, der erforderlich ist, um diese kontinuierlich an die sich ständig weiterentwickelnden Krankenhaus-, Behandlungs- und patientenspezifischen Anforderungen anzupassen.

Das Hauptforschungsziel dieser Arbeit ist die Entwicklung eines kollaborativen, rein metamodellbasierten Adaptive Case Management for Integrated Care (ACM4IC) Ansatzes. Wir adressieren die drei Herausforderungen der integrierten Versorgung mit i) adaptivem Fallmanagement, das die Modellierung behandlungsspezifischer Fallvorlagen ermöglicht und bei Bedarf patientenzentrierte Laufzeitanpassungen ermöglicht, ii) rein metamodellbasierten Integrationsmustern, die behandlungs- und krankenhausspezifische Anpassungen ermöglichen und iii) nahtlos integrierten, fallbezogenen Kollaborationsfunktionen.

Zunächst werden die Anforderungen aus der Literatur zu den drei Herausforderungen der integrierten Versorgung abgeleitet. Anschließend werden diese Anforderungen im Konzeptentwurf in ein ganzheitliches Metamodell integriert, während eine prototypische Implementierung eine Bewertung in der Praxis ermöglicht. Wir haben unseren Ansatz im Rahmen eines internationalen Projekts zur integrierten Versorgung während Implementierungsstudien an drei Klinikstandorten in Groningen, Tel Aviv und Lleida in den Niederlanden, Israel und Spanien mit jeweils zwei Fallstudien evaluiert. Unser ACM4IC-Ansatz wurde rein metamodellbasiert in die Projektarchitektur integriert und in einer produktiven Cloudumgebung bereitgestellt. Fallvorlagen wurden unter Berücksichtigung praktischer klinikspezifischer Anforderungen für die Fallstudien modelliert und evolutionär verbessert. Während der Implementierungsstudien werden 44 Fallvorlagenversionen für den produktiven klinischen Einsatz bereitgestellt. Der Prozess der Case-Template Modellierung wurde über ca. 20 Monate analysiert und über Fallstudien hinweg verglichen. Insgesamt wurden 232 Patienten innerhalb von etwa einem Jahr behandelt. Wir haben das Verhalten der Fallbearbeitung analysiert und über alle Fallstudien hinweg verglichen.

Diese Arbeit führt zu mehreren Beiträgen: Erstens sind die abgeleiteten Anforderungen für die Entwicklung verwandter Softwareansätze wiederverwendbar. Zweitens liefert das konzeptionelle Design bzw. die prototypische Implementierung, einschließlich der Diskussionen über herausfordernde Aspekte, Erkenntnisse zur Umsetzung dieser Anforderungen. Drittens beweist die Prototyp-Evaluierung die praktische Anwendbarkeit in einer realen integrierten Pflegeumgebung jenseits eines kontrollierten Laborversuchs. Praktische Erkenntnisse verbesserten evolutionär unser anfängliches konzeptionelles Design und die prototypische Implementierung. Darüber hinaus zeigt die kritische Reflexion Verbesserungsvorschläge auf und leitet weitere Forschungsmöglichkeiten ab.

Abstract

Nowadays, integrated care is widely acknowledged as a promising approach for patient-centric treatments, while adequate software support for integrated care is still missing. Challenges are i) highly context-dependent unpredictable treatments, ii) semantic information exchange and system interoperability respectively, and iii) coordination across multiple organizations and different roles. Paper-based approaches prevent sufficient collaboration across care professionals while traditional implementations combining a workflow engine with a hard-wired frontend are inadequate, considering the effort required to continuously adapt to the evolutionarily changing evolving hospital-, treatment-, and patient-specific requirements.

The main research goal of this thesis is to develop a collaborative, purely meta-model-based Adaptive Case Management for Integrated Care (ACM4IC) approach. We address the three integrated care challenges with i) adaptive case management, which enables modeling treatment-specific case templates and allows patient-centric run-time adaptions where required, ii) purely meta-model-based integration patterns that enable treatment-, and hospital-specific customizations, and iii) seamlessly incorporated case-based collaboration capabilities.

At first, the requirements are derived from the literature addressing the three integrated care challenges. Subsequently, the conceptual design incorporates those requirements into a holistic meta-model, while a prototypical implementation enables an evaluation in practice. We evaluated our approach within an international integrated care project during implementation studies on three clinic locations in Groningen, Tel Aviv, and Lleida, located in the Netherlands, Israel, and Spain with two case studies each. Our ACM4IC approach was integrated purely meta-model-based into the project architecture and deployed in a productive cloud environment. Case templates were modeled considering practical clinic-specific requirements for the case studies and were evolutionarily improved. During the implementation studies, 44 case template versions were deployed for productive clinical usage. The case-template-modeling process was analyzed over approximately 20 months and compared across the case studies. In total, 232 patients were treated within about one year. We analyzed the case execution behavior and compared it across all case studies.

This thesis results in multiple contributions: Firstly, the derived requirements are reusable for developing related software approaches. Secondly, the conceptual design, respectively the prototypical implementation, including the discussions regarding challenging aspects, provide insights for the implementation of those requirements. Thirdly, the prototype evaluation proves the practical applicability in a real-world integrated care environment beyond a controlled laboratory experiment. Practical findings evolutionarily improved our initial conceptual design and prototypical implementation. Additionally, the critical reflection indicates suggestions for improvement and deduces further research opportunities.

Acknowledgment

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Dr. Florian Matthes for his continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me throughout the entire time dedicated to the research and writing this thesis. Further, I would like to express my sincere gratitude to my second advisor Prof. Dr. Martin Bichler.

I would like to express a special appreciation to the Personalised Connected Care for Complex Chronic Patients project that received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement n° 689802). Significant contributions of the project consortium enriched the prototypical implementation and enabled scientific contributions. I would especially like to thank Eloisa Vargiu Ph.D., Jak Kelly, Juan Manuel Fernández, Mauricio Gonzales, Kitiara Prunera-Moreda, Ilya Kurnosenkov, Gerard Torres Cortada Ph.D., Jordi de Batlle, and Felip Miralles Ph.D. for the close interdisciplinary and fruitful cooperation.

The chair of Software Engineering for Business Information Systems at the Technische Universität München provided an excellent environment for my research. Therefore, I would like to thank my colleagues for the great and fruitful collaboration; especially those of my colleagues who directly or indirectly contributed to this thesis: Patrick Holl for his modeling contribution, Manoj Mahabaleshwar and Dr. Thomas Reschenhofer for the collaborative technical support regarding the Hybrid Wiki and End-Users Analytics implementation and Adrian Hernandez-Mendez for the collaborative scientific contribution regarding model-based systems, Dr. Sven-Volker Rehm for the collaboration regarding system adaptability in chronological care management, Dr. Matheus Hauder and Yolanda Gil Ph.D. for the joint scientific publications in the workflow management field. I would like to thank the student Simon Bönisch who contributed parts of the technical infrastructure to analyze the executed cases.

Finally, and most importantly, I want to thank my family for their continuous support over the recent demanding years. Last, but not least, I would like to thank my partner for her support, patience, and encouragement during that challenging time.

Garching b. München, 9.12.2019

F. Michel

Felix Michel

Table of Contents

List of Figures

List of Tables

List of Listings

CHAPTER 1

Motivation and Introduction

1.1. Problem Description

The demographic change in Europe leads to an aging population. In fact, the share of people older than 65 years will rise significantly from 17.4 percent in 2010 to 25.6 percent in 2030 and at a slower rate rise to 29.5 percent until 2060 (European Commission, 2014, p. 18). In the aging society of Germany, the risk for two or more simultaneously occurring chronic diseases, the so-called multimorbidity, increases (Robert Koch-Institute, 2016). Within the age group 65-75, multimorbidity affects 68 percent of the men and 76 percent of the women, while in the age group 75-79, multimorbidity affects 74 percent and 82 percent respectively (Robert Koch-Institute, 2016). In conjunction with the demographic change, the disease spectrum is shifting in relation to age, so chronic diseases will occur more frequently, which has implications on the design of future healthcare solutions (Robert Koch-Institute, 2015).

Additionally, it is broadly acknowledged that elderly, chronically ill patients may regularly suffer unexpected emergency room visits or hospital admissions caused by medical or social circumstances triggering a negative effect. A common elderly patient with at least two chronic diseases who typically relies on excessive healthcare resources is considered as Complex Chronic Patient (CCP). While the share of CCPs within society is relatively low, they are responsible for a disproportionately high fraction of hospital admissions. (Vargiu et al., 2017)

Traditionally, a specialist involved in the patient's treatment is responsible for exactly one of the multiple diseases (Tinetti et al., 2012). Uncoordinated simultaneous treatments imply a high uncertainty regarding the benefits and harm caused by side effects (Tinetti et al., 2012). Integrated care is a promising holistic approach to improve the treatment of CCPs. The different aspects of the multifaceted term *integrated care* are elaborated in the report of the World

Health Organization (2016). While most definitions are vague, we consider integrated care precisely as an organizational model which "will facilitate collaboration and communication among healthcare professionals, patients and their carers through integrated technological solutions in which the patients play a central role [...] and empower patients for self-management, by providing them recommendations and suggestions according to continuous monitoring of their activities" (Vargiu et al., 2017). Establishing integrated care in practice is a multidimensional challenge, hence we focus on the most important aspects relevant concerning the system design. The following major high-level challenges are commonly known:

Highly context-dependent, unpredictable treatments Complex interactions between conditions, treatments, and medication prevent deterministic decisions (Hollingsworth, 2010). In clinical practice, treatment task flows are often context-dependent and do not follow clear completion strategies as a result of interruptions, uncertainties, and strongly collaborative characteristics (Horsky et al., 2005). Latest medical knowledge resulting from clinical research must be incorporated into clinical routines (Garde and Knaup, 2006), thus requiring easily adaptable systems.

Semantic information exchange and system interoperability The heterogeneity of healthcare services and the resulting complex system landscape lead to challenges regarding system interoperability, that is, the semantical information sharing required for efficient health care (Garde and Knaup, 2006). The progressive specialization on disease-focused medicine additionally leads to service fragmentation that threatens holistic care approaches (Valentijn et al., 2013).

Coordination across multiple organizations and different roles Patient treatment is characterized as a knowledge-intensive process that typically requires communication between involved professionals across organizational boundaries, whereas neglected communication leads to increased consumption of health resources and might negatively impact the patient's health (Garde and Knaup, 2006). Similarly, speedy and clear communication are acknowledged as a critical treatment factors (Horsky et al., 2005).

While the need for integrated care is wildly acknowledged, adequate support for integrated care is still missing. Integrated care involves multiple professional roles and is a highly context-dependent, knowledge-intensive endeavor. The widespread Business Process Model and Notation (BPMN) specified by the Object Management Group (2007, 2014a) is suitable for processes predictable at design time. In clinical practice, BPMN is applied to administrative processes but is not suitable for the knowledge-intensive care activities (Hollingsworth, 2010). In contrast to the classical workflow management, Adaptive Case Management (ACM) is a holistic approach that enables flexible process adaptations during run-time (Fischer, 2011, p. 11), which is suitable to handle knowledge-intensive care activities. The Case Management Model and Notation (CMMN) specified by the Object Management Group (2014b, 2016) enables modeling templates for those unpredictable processes with certain limitations (Kurz et al., 2015).

Considering the continuously evolving hospital- and treatment-specific requirements, traditional custom implementations or adaptations are lengthy and costly. Combining an existing generic ACM engine with a traditional hard-wired frontend implementation reduces the effort and enables customized solutions. Those combined approaches are suitable for enterprises considering a large number of cases to amortize the custom hard-wired implementation, but inadequate for integrated care. For the sake of completeness, we want to empathize that classical paper-based approaches without tool support hinder adequate information sharing and prevent a close collaboration among care professionals from multiple organizations to provide patient-centered care.

Figure 1.1 schematically visualizes the integrated care environment context without integrated tool support for care professionals on the left and with integrated generic tool support for care professionals on the right. As elaborated above, we are aware that semi-hard-wired inadequate approaches might exist, but we intend to elaborate on the conceptual gap for generic tool support. Care professionals involved within an integrated care approach are traditionally distributed across multiple organizations. The organizational structure primarily depends on national health care systems. In Israel, specialist doctors and hospital staff are normally co-located within one organization. In contrast, the care professional roles in Spain are typically separated into different organizations, while particular co-locations occur. During patient treatment, medical questionnaires are used to evaluate the patient's status. Uncoordinated care might lead to redundant evaluations leading to slightly different results and thus to possibly diverse conclusions. Simultaneous treatments that are frequently required for chronical patients might cause undesired negative side-effects when uncoordinated. Therefore, integrated care must address semantic information exchange across organizational boundaries and facilitate the communication and coordination between care professionals to enable a context-dependent patient-oriented treatment. Integrated care primarily depends on aggregated patient information to provide a comprehensive context for care professionals. Enabling integrated care tailored to the hospital- and treatment-specific needs requires an Adaptive Case Management for Integrated Care (ACM4IC) approach that can easily be adapted without programming effort.

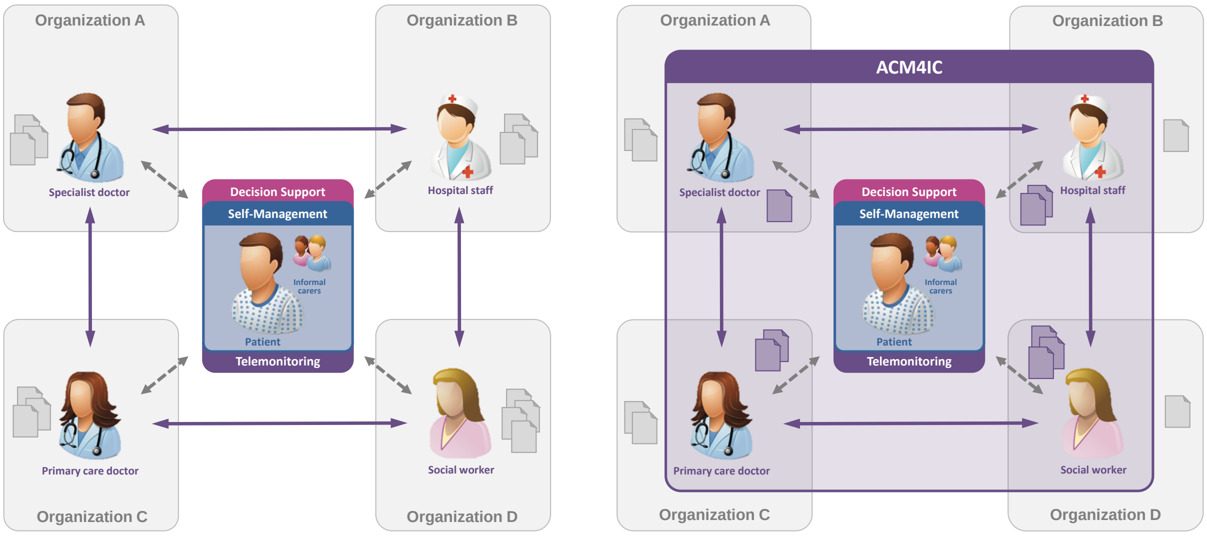


Figure 1.1.: Schematic problem visualization. The integrated care context without tool support to orchestrate the patient-centered treatment across organizational boundaries and with desired integrated tool support illustrated on the left and right accordingly. Adapted and extended from CONNECARE Consortium (2016).

1.2. Research Questions

The main objective of this thesis is derived from the problem description and subdivided into precise research questions. The key contribution is intended to narrow the gap towards supporting a holistic patient-centered treatment process with a purely meta-model-based approach and leads to our research hypothesis:

Research Hypothesis: Adaptive Case Management for Integrated Care (ACM4IC) will empower care professionals with a collaborative, purely meta-model-based software solution customizable to hospital-, treatment-, and patient-specific needs, to enable patient-centric treatments across organizational boundaries.

Subsequently, the research hypothesis is subdivided into specific research questions structuring the overall objective. The first research question addresses the requirements which are reusable to implement related software approaches:

RQ1: What are the key requirements for ACM4IC?

Primarily depending on the three high-level challenges emphasized within the problem description (cf. Section 1.1), the requirements for ACM4IC are derived from the literature in Chapter 3. The three cosponsoring high-level requirements are further decomposed into more specific requirements to aid the conceptual design, which leads to the following research question:

RQ2: What are the key aspects of the ACM4IC conceptual design?

The conceptual design is decomposed into eight conceptual architectural layers to encapsulate functionality in Section 4.1. Based on the layered architecture from the Hybrid Wiki approach, three additional layers are designed, while underling layers are slightly extended. Structured and colored according to the conceptual layers, the resulting conceptual unified meta-model combines data-modeling, adaptive process modeling, and seamless integrated communication and coordination capabilities as elaborated in Section 4.2. The initially presented Hybrid Wiki meta-model (Matthes et al., 2011; cf. Section 2.2.2), extended with further analytical capabilities (Reschenhofer and Matthes, 2016b; cf. Section 2.2.5), serves as a conceptual foundation regarding the data modeling capabilities. The CMMN specification version 1.1 (Object Management Group, 2016; cf. Section 2.1.2) is used as a reference for process modeling capabilities whereas inspired by previous work (Michel et al., 2015b; cf. Section 2.2.3; Hauder, 2016; cf. Section 2.2.4). An adaptive case management approach comes to life with the meta-model-related execution semantics described in Section 4.3. Primary conceptual design challenges are highlighted in Section 4.4. We conclude the conceptual design with a summary of supported CMMN elements in Section 4.5 and a to matrix indicate which conceptual layer addressed which requirement in Section 4.6. However, care professionals as end-users require a valuable tool considering those concepts, which leads to the following research question:

RQ3: What are the key aspects of the ACM4IC prototypical implementation?

Care professionals are considered as the primary stakeholders. Therefore, the end-user interface must support care professionals during their daily work, which is inspired by the design principles (Michel et al., 2015b; cf. Section 2.2.3). A primary concern is to continue the purely meta-model-based approach resulting from the conceptual design (cf. Chapter 4) to enable declarative contextual customization. Multiple iterative evaluations with care professionals led to the end-user interface features presented in Section 5.1. A meta-model-based end-user interface becomes usable with the corresponding case templates. A comprehensive purely meta-model-based approach as ACM4IC hence requires expressive grammar to enable modeling full-stack case templates containing model elements from multiple conceptual layers. The knowledge-intensive characteristics of ACM require run-time flexibility, likely leading to unpredictable execution paths that may result in unexpected run-time errors. Therefore, modelers must be supported to declare test executions to enable testing at least certain execution paths. The resulting case template grammar reference is presented in Section 5.2, whereas modeling best practice principles are elaborated in Section 5.3, and the import workflow to transforming declared case templates into executable case templates is detailed in Section 5.4. Considering the importance of the semantic information exchange and the system interoperability respectively, a generic API design is required that is sufficient for the professional end-user interface and third-party integration, as elaborated in Section 5.5. During the prototypical implementation, multiple technical challenges occurred, the fundamental ones of which are highlighted in Section 5.6. The supported requirements are summarized with the help of end-user interface features in Section 5.7. Finally, the last research question leads to the practical applicability outside of a controlled laboratory environment:

RQ4: What are the experiences using ACM4IC in practice?

Practical experience is most crucial to evaluating the applicability of a pure meta-model-based ACM4IC approach. In the context of a European integrated care project, the ACM4IC prototype is used in practice beyond a controlled laboratory environment. The project context and applied case studies within multiple countries are elaborated in Section 6.2. The project's system architecture with the integrated ACM4IC approach and the related productive system deployment are presented in Sections 6.3 and 6.4. Assuming sufficient expressive power, the usability of a purely meta-model-based system depends primarily on case templates as described in Section 6.5. The analysis of the case execution generates insights regarding the actual usage behavior; those are presented Section in 6.6. Finally, the practical experience is summarized in Section 6.7.

1.3. Research Design

We adapted the design science framework for Information Systems Research (ISR) which was originally proposed by Hevner et al. (2004). Figure 1.2 illustrates the adapted conceptual framework applied for the context of this thesis. It primarily consist of the following principles:

- **Environment:** Describes the problem space of interest. The environment considers that *people* with a medical or information technology background and several roles, such as clinical professionals, primary care professionals, case template modelers, developers, and architects are involved. The *clinical organizations* providing the medical use cases and we collaborated with *technological organizations* to integrate our prototypical solution into a real-world scenario. The organizations are internationally distributed hospitals, universities, and enterprises. Furthermore, information *technology* exists in hospitals and technical organizations.

- **Knowledge Base:** Represents prior results gained from ISR that might be used as a foundation or methodology. Our knowledge base primarily consists of the Adaptive Case Management (ACM) concept described by Swenson (2010) and Fischer (2011) in combination with the Case Management Model and Notation (CMMN) version 1.1 specified by the Object Management Group (2016). Model-based information systems as summarized by Hesse and Mayr (2008) are another relevant design paradigm.

- **IS Research:** Is accomplished based on the *business needs* derived from the *environment* and the *applicable knowledge* from the *knowledge base*. The iterative artifact creation with alternating development and evaluation leads to potential additions to the knowledge base and to an application within the environment. Our holistic, collaborative adaptive case management prototype is accessed by the clinical organization with actual use cases to refine the artifact within the next iteration.

- **Business Needs:** The environment setting creates *business needs* influencing the ISR. The heterogeneity within our environment caused by the organizational, legal, and technical differences lead to various *business needs* that must be coverable through model adjustment. On the other side, the high heterogeneity increases the transferability of our conceptual design and prototypical implementation.

- **Applicable Knowledge:** The knowledge base provides *applicable knowledge* influencing the ISR. In our context, ACM principles described by Swenson (2010) and Fischer (2011) are applied and a subset of the CMMN notation version 1.1, as specified by Object Management Group (2016), is prototypically implemented and extend according to business needs. The iterative artifact creation process provides additions to the knowledge base that are disseminated as publications.

Design science is an iterative problem-solving approach with the described fundamental principles. We applied the seven design science research guidelines derived from the principles according to Hevner et al. (2004) as follows:

Guideline 1: Design as an Artifact During this thesis, we develop a collaborative, purely model-based ACM4IC approach as the primary IT artifact regarding design science. The conceptual design is elaborated in Chapter 4 and highlights of prototypical implementation in Chapter 5 respectively. Consequently, the prototype accordingly serves as an instantiated IT artifact and presents the proof of concept.

Guideline 2: Problem Relevance Emphasized in Section 1.1, the demographic change in Europe leads to an aging population (European Commission, 2014, p. 18). Above the age of 65 years, multiple chronically diseases occur frequently (Robert Koch-Institute, 2016). An elderly patient with multiple chronic diseases typically consumes disproportionately high share of healthcare resources. Uncoordinated, concurrent treatments imply a high uncertainty regarding the benefits and harm caused by side effects (Tinetti et al., 2012). Integrated care is a promising holistic, patient-centered approach to improve the treatment of those patients. While the need for integrated care is wildly acknowledged, adequate support for integrated care is still missing. Considering the continuously evolving hospital- and treatment-specific requirements, traditional custom implementations or adaptations are inadequate.

Guideline 3: Design Evaluation As elaborated in Section 6.1, our approach was evaluated with multiple iterations that have been used to continuously improve our approach. Overall, Chapter 6 presents the final evaluation results including an introduction of the project used for evaluation purpose in Section 6.2, the architecture to integrate our approach into an integrated care environment in Section 6.3, presents the modeled case templates in Section 6.5.1, and the case template execution behavior is described in Section 6.6.

Guideline 4: Research Contribution The first contribution is the conceptual framework for the ACM4IC approach that includes the classification of existing tools elaborated in Chapter 2, and crucial requirements derived from the literature to support such an approach are presented in Chapter 3. Second, the related conceptual design elaborated in Chapter 4 serves as the primary research contribution of this thesis. Consequently, the prototypical implementation described in Chapter 5 serves as a proof of concept, whereas the evaluation presented in Chapter 6 proves the practical applicability as the third contribution.

Guideline 5: Research Rigor Our conceptual foundation and related work are presented in Chapter 2. The Adaptive Case Management paradigm applicable for knowledge-intensive processes described by Swenson (2010); Fischer (2011) and the CMMN notation to express knowledge-intensive processes specified by the Object Management Group (2016) provide the theoretical foundation. We classified existing tools regarding integrated care capabilities and summarized the related work chronologically. Our approach is based on the Hybrid Wiki (Matthes et al., 2011), a data-driven meta-model platform, inspired by Darwin (Hauder, 2016), a lightweight approach empowering end-users to structure knowledge-intensive processes and incorporates the End-User Analytical capabilities presented by Reschenhofer (2017).

Guideline 6: Design as a Search The iteratively developed IT artifact has been continuously evaluated and improved with clinical use cases. Our primary artifact, the case execution engine including collaboration capabilities, required comprehensive models to achieve rational and constructive evaluation results. Therefore, we applied a two-dimensional nested system and model evaluation lifecycle as elaborated in Section 6.1. The system design-time phase evaluation typically depends on the system run-time phase that includes the model design-time phase and the model run-time phase. Within one system design-time iteration, multiple model iterations are typically performed. The purely artifact-centric iterations are wrapped and enhanced long-running PDSA cycles for strategic management decisions, influencing the artifact creation. This approach ensures a continuously improving artifact that is aligned with the business needs.

Guideline 7: Communication of Research Finally, we published the resulting prototype, use cases, and selected preliminary results on conferences focusing on Information Systems (IS) and Integrated Care. Figure 1.3 summarizes all scientific publications related to the relevant design science artifacts. Additionally, informal information exchange and discussions with other researchers, especially focusing on model-based information systems, adaptive case management, and integrated care took place.

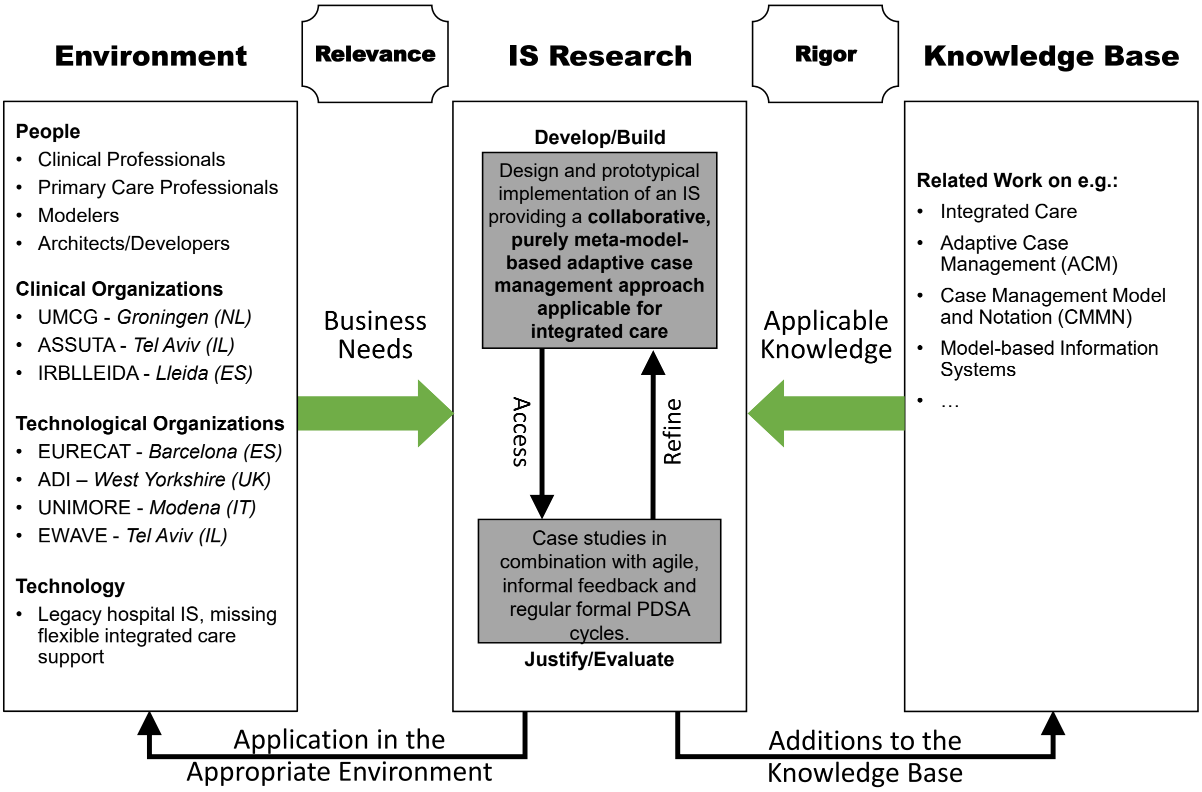

Environment
Relevance
IS Research
Rigor
Knowledge Base
People
• Clinical Professionals
• Primary Care Professionals
• Modelers
• Architects/Developers
Clinical Organizations
• UMCG - Groningen (NL)
• ASSUTA - Tel Aviv (IL)
• IRBLLEIDA - Lleida (ES)
Technological Organizations
• EURECAT - Barcelona (ES)
• ADI – West Yorkshire (UK)
• UNIMORE - Modena (IT)
• EWAVE - Tel Aviv (IL)
Technology
• Legacy hospital IS, missing flexible integrated care support
Business Needs
Develop/Build
Design and prototypical implementation of an IS providing a collaborative, purely meta-model-based adaptive case management approach applicable for integrated care
Access
Refine
Case studies in combination with agile, informal feedback and regular formal PDSA cycles.
Justify/Evaluate
Applicable Knowledge
Related Work on e.g.:
• Integrated Care
• Adaptive Case Management (ACM)
• Case Management Model and Notation (CMMN)
• Model-based Information Systems
• …
Application in the Appropriate Environment
Additions to the Knowledge Base

Figure 1.2.: The information systems research framework from Hevner et al. (2004) adapted to the research design of this thesis.

1.4. Contributions of this Thesis

We summarized the main contribution of this thesis in Figure 1.3. According to the thesis chapters, the research results and research artifacts are presented with boxes highlighted in green in the center. The research questions are associated with the related chapters and the related publications are listed accordingly.

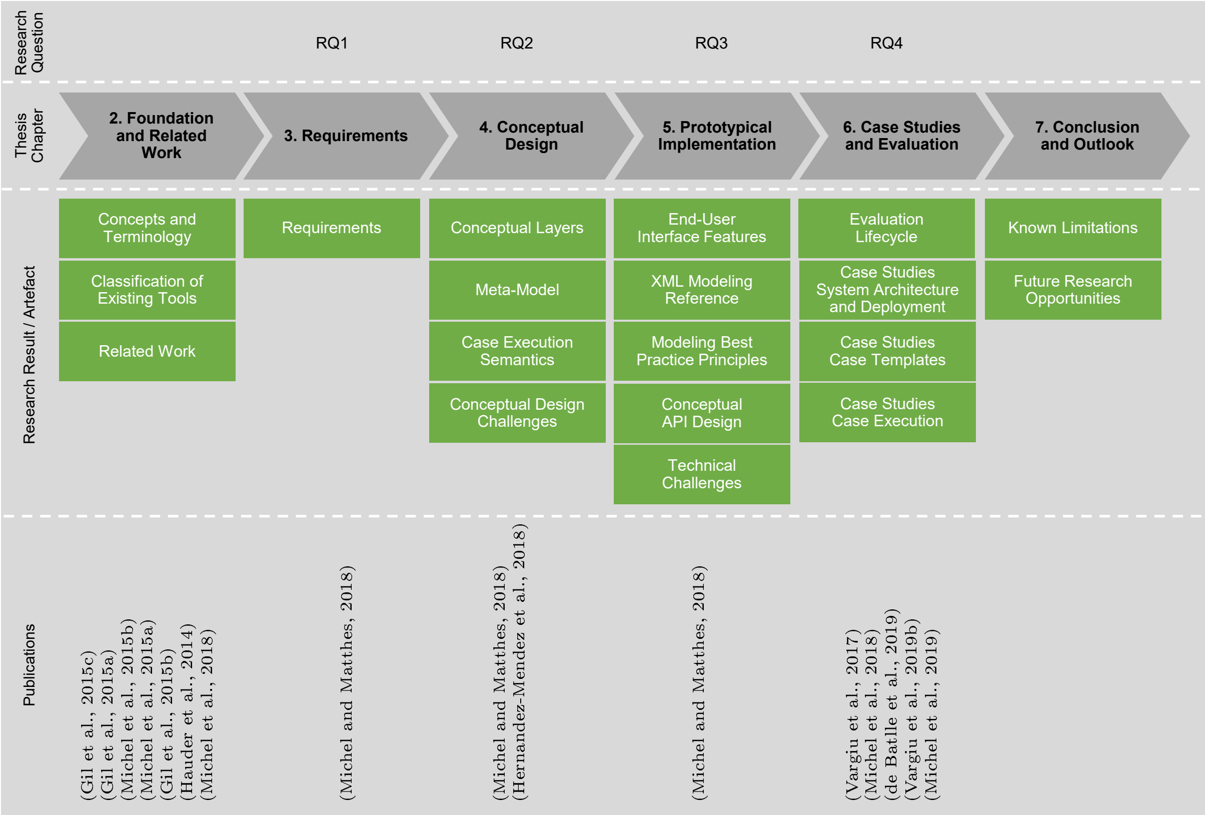


Figure 1.3.: The main contributions of this thesis.

The first contribution is a comprehensive introduction which includes the presentation of the Adaptive Case Management (ACM) concept, the related Case Management and Model Notation (CMMN) terminology, a classification existing tools and related work as presented in Chapter 2. Requirements for an ACM4IC approach derived from the literature are elaborated in Chapter 3. Those generic requirements are reusable for related software approaches in the integrated care context.

The second contribution includes the conceptual design and the prototypical implementation of the collaborative purely meta-model-based ACM4IC approach. The conceptual design presented in Chapter 4 addresses the requirements and contributes the extension of the conceptual architectural layers, the related meta-model, and the case execution semantics. It also highlights crucial conceptual design challenges. Accordingly, derived from the conceptual design, the prototypical implementation presented in Chapter 5 contributes end-user interface features, the XML case template modeling reference including samples, the best practice modeling principles, and the conceptual API design while highlighting the technical challenges.

The third contribution is the prototype evaluation with case studies in a real-world environment as presented in Chapter 6. A comprehensive evaluation of a meta-model-based system must include the case templates. Therefore, a multidimensional evaluation-lifecycle is presented and applied. The evaluation is performed with two conceptually different integrated care case studies in three international hospitals within a European Horizon 2020 project. Hence, the integration architecture and the deployment within the integrated care context constitute an essential contribution. The case study and site-specific case templates, including the analyzed case execution, represent an additional contribution. In conclusion, the evaluation demonstrates the applicability within the integrated care context.

1.5. Outline of the Thesis

The problem statement elaborated in Section 1.1 and the research methodology presented in Section 1.3 are lead to the outline of this thesis. Accordingly, the thesis is subdivided into the following seven chapters:

Chapter 1: Motivation and Introduction motivates the thesis and provides a comprehensive introduction including the formulated the problem description, the derived research questions, the applied research design according to Hevner et al. (2004) and the main contribution of this thesis with the crucial publications.

Chapter 2: Foundation and Related Work presents fundamental work relevant for the thesis and related work our approach builds on. Therefore, the Adaptive Case Management concept is elaborated and defined, the related most common CMMN elements are explained, and state-of-the-art tools are classified according to their capabilities.

Chapter 3: Requirements crucial for an ACM4IC approach are derived from the literature and categorized into three dimensions. The requirements are relevant for the conceptual design and the prototypical implementation. Furthermore, those requirements are valuable for related software approaches in the integrated care environment.

Chapter 4: Conceptual Design elaborates various aspects of the design for an ACM4IC approach. It covers the conceptual architectural layers, the conceptual meta-model with their elements, the execution semantics, and critical design challenges with related conceptual solutions. Finally, supported modeling elements according to the CMMN notation are presented.

Chapter 5: Prototypical Implementation presents the implementation of the conceptual design. Conceptually crucial end-user interface features are explained. A case template definition reference including sample declarations is provided and modeling best-practice principles are presented. Furthermore, the conceptual API design is elaborated, and several fundamental technical challenges are illustrated.

Chapter 6: Case Studies and Evaluation are applied within three different internationally distributed hospitals. The iterative evaluation process with a system and model dimension is presented. The integrated care project used for the evaluation purpose is introduced briefly, including the conceptual integration architecture. Case study-related modeling artifacts are elaborated, and the case execution behavior is analyzed.

Chapter 7 Conclusion and Outlook presents a comprehensive thesis summary highlighting the crucial aspects. Furthermore, the prototypical solution is critically reflected and known limitations are illustrated. Similarly, the evaluation is critically reflected and corresponding limitations are elaborated. Finally, promising further research opportunities are illustrated.

CHAPTER 2

Foundation and Related Work

2.1. Foundation

This section provides the foundations, including adaptive case management terminology and related clinical practice in Section 2.1.1. The corresponding case management model and notation is presented in Section 2.1.2 accordingly. Additionally, a classification of existing tools according to crucial capabilities is presented in Section 2.1.3.

2.1.1. Adaptive Case Management (ACM)

This adaptive case management foundation section originally appeared in Michel et al. (2019) as presented in the following. Software engineering and business process management literature discuss process adaptations under the term case management, defining two categories of workflow management application systems (van der Aalst et al., 2005; Reichert and Reijers, 2016). Production case management (PCM) refers to processes that are defined by software engineers at design-time and principally remain stable while serving for a particular domain or problem (Swenson, 2012). Adaptive case management (ACM) on the contrary refers to situations "where the path of execution cannot be predetermined in advance of execution; where human judgment is required to determine how the end goal can be achieved; and where the state of a case can be altered by external out-of-band events" (White, 2009).[1]

[1] The entire section 2.1.1 has been published as presented in Michel et al. (2019).

ACM includes knowledge workers into system adaptations at run-time (Hauder et al., 2014). This system capability to adapt processes at run-time enables a quicker response to organizational or routine changes, to master unpredictable situations in processes, facilitating continuous service provision as well as learning effects (Swenson, 2010; Marin et al., 2016). However, this requires systems to provide appropriate execution environments that enable knowledge workers without programming and modelling expertise to modify processes on their own during run-time (Marin et al., 2016). Consequently, respective design requirements for execution environments have been defined along classifications such as data integration, knowledge worker empowerment, authorization and role management, knowledge storage and extraction, and more formal definitions of adaptability, routines and further factors (Hauder et al., 2014).

Research on HIT adoption has emphasized that "organizational differences [...] may be particularly salient in the healthcare setting" (Avgar et al., 2018) while the need for integrating process stakeholders and variations in work practices seem to be "especially pronounced" (Avgar et al., 2018; Hitt and Tambe, 2016). With regard to clinical practice, particularly in context of care for chronic diseases, ACM thus becomes increasingly relevant to configure care processes' fit to the individual patient (Agarwal et al., 2010; Fichman et al., 2011; Lin et al., 2017). This comprises tasks such as "prevention and detection of acute events through continuous monitoring and assessment; patient [...] behaviour modification [...]; specialized treatment plans coordinated by disease experts; and preserved continuity of care across diverse patient care settings" (Ouwens et al., 2005; Ferguson and Weinberger, 1998).

Previous research in integrated care contexts has identified limitations of current ACM HIS capabilities such as "[...] lack of contextualization and poor dynamic adaptation to changes" and request "(ad-hoc) modifications to (process models) implemented in the process of (HIS) development" (Cano et al., 2015). Current recommendations for ACM implementation focused on chronic care management suggest treatment templates that include steps such as case identification; case evaluation/patient characterization; personalized work plan definition/execution/follow-up; event handling; and discharge (Cano et al., 2017; Swenson, 2010).

DEFINITION: Adaptive Case Management

The aim of adaptive case management "[...] is to manage all of the work needed to handle a given case, regardless of type, such as automated work, manual work done by people, ad-hoc work, content-intensive work, etc. The result, if fully embraced, is to create a living, breathing network of work that reflects the dynamism of the environment in which it is performed." (Burns, 2011, p. 17)

2.1.2. Case Management Model and Notation (CMMN)

The Case Management Model and Notation (CMMN) is published and maintained by the Object Management Group (OMG), an international consortium that specifies and maintains modeling standards. The CMMN specification version 1.0 was published in May 2014 and an updated version 1.1 was published in December 2016 (Object Management Group, 2014b, 2016). The specification contains a graphical notation and defines an exchange format to enable interoperability between different tool vendors. Our practical experience shows that the interoperability between different tool vendors is currently still limited. In the following, we will focus on summarizing the CMMN core concepts and their expressiveness defined by the Object Management Group (2016).

Case Management is typically applied to a knowledge-intensive process. In the healthcare domain, a medical diagnoses and the related treatment is a classical example. A patient treatment does not allow to completely determine a case-flow at design-time, but certain steps are similar for every patient treatment. Typically, several ad-hoc decisions and treatment adaptions performed by a case-worker are needed. De Man (2009) states that the case-worker as a human factor should be predominant and may significantly influence the case process. Therefore, CMMN supports modeling during the design-time phase and rolling planning during the run-time phase, as illustrated in Figure 2.1. The CMMN specification distinguishes between the `plan items` that should be executed during the run-time phase and `discretionary items` that are not executed automatically during the run-time phase. To execute discretionary items, a case-worker must perform run-time planning to decide whether a discretionary item is actually useful in the current case context. Over time, the decisions performed during the ad-hoc run-time planning can lead to case model improvements. Frequently used fragments may transform into plan items.

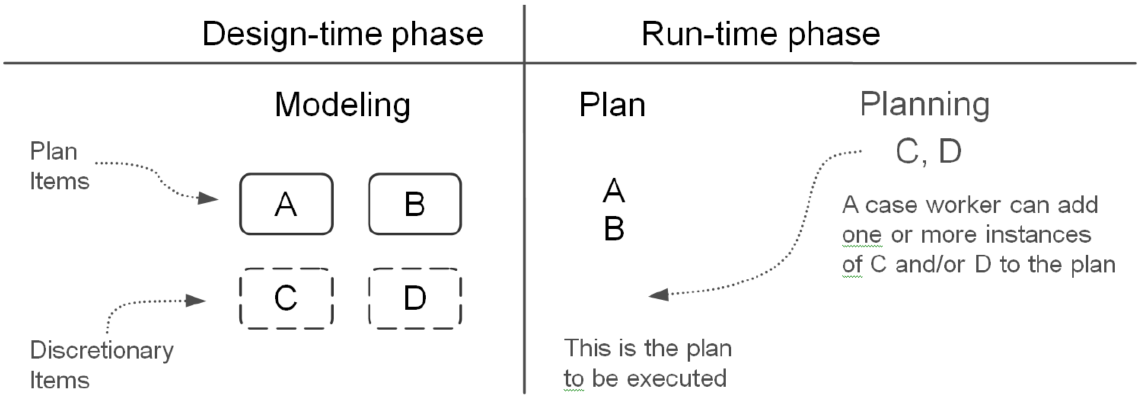

Design-time phase
Run-time phase
Modeling
Plan
Planning
Plan Items
A
B
C
D
Discretionary Items
A
B
C, D
A case worker can add one or more instances of C and/or D to the plan
This is the plan to be executed

Figure 2.1.: Design-time modeling and run-time planning in CMMN (Object Management Group, 2016).

A CasePlanModel is used as a container for a knowledge-intensive process, comparable with a template that contains some predefined fragments and allows flexible adjustments during the run-time, depending on the model. A case is notated with a rectangle and small trapeze on top containing the case name, as illustrated in Figure 2.2. Within the rectangle, a typical case contains many CMMN elements further specifying the case.

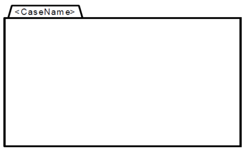


Figure 2.2.: CasePlanModel notated in CMMN (Object Management Group, 2016).

A Stage is a structuring element to group multiple CMMN elements into a container. Graphically, a stage is notated with an octagon visually comparable with a rectangle having angled corners, as illustrated in Figure 2.3. Planned stages represented with a solid border and discretionary stages with a dashed border. Stages are either represented as extended, showing all containing CMMN elements, which is indicated with the minus icon on the bottom, or as collapsed, not showing any containing CMMN elements, which is indicated with a plus icon on the bottom. Stages can be nested to express grouped subprocesses.

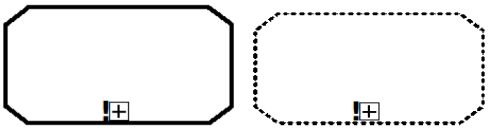


Figure 2.3.: Planned and discretionary Stage notated in CMMN (Object Management Group, 2016).

A Task is visually notated with a rectangle with rounded corners and containing the task's label in its center. Planned tasks represented with a solid border and discretionary tasks with a dashed border. Different task types are distinguished with an icon on the top left corner. Blocking human tasks are visualized with a person icon, non blocking human tasks are represented with a hand icon and process task are visualized with a chevron symbol, as illustrated in Figure 2.4.

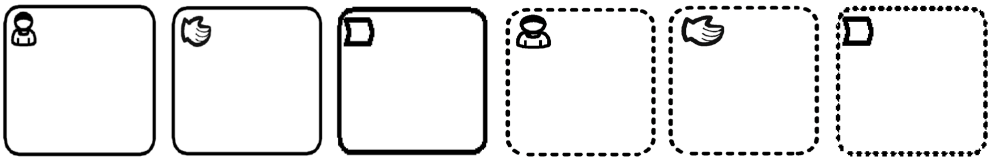


Figure 2.4.: Planned and discretionary blocking HumanTask, non-blocking HumanTask and ProcessTask notated in CMMN (Object Management Group, 2016).

A Milestone is graphically notated with a rectangle in which the corner radius is precisely half the shape height, as illustrated in Figure 2.5. The milestone name is placed centered.

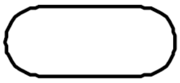


Figure 2.5.: Milestone notated in CMMN (Object Management Group, 2016).

An EventListener is notated as double line circle contains a symbol indicating the dedicated event type. Figure 2.6 illustrates a TimedEventListener, indicated with a clock symbol and a UserEventListener, indicated with a human person symbol.

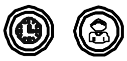


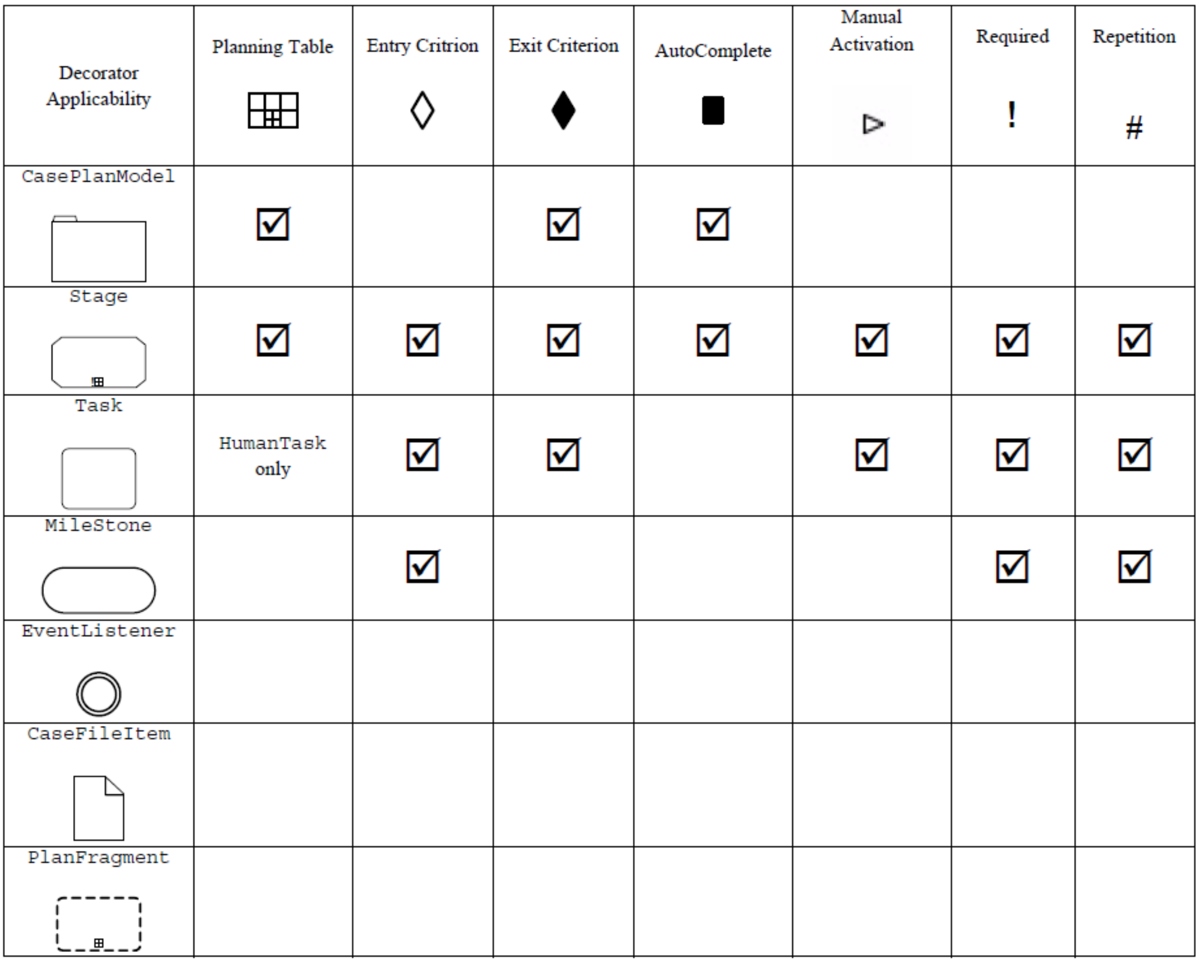
Figure 2.6.: Timed event listener and User event listener notated in CMMN (Object Management Group, 2016).

A CaseFile represents all case information and typically contains many CaseFileItems. A CaseFile is a generic content object that can represent any type of data structure such as a folder, document, or only a reference. Each CaseFile has meta-data fields such as the name, a reference to the parent element, a reference to the children, a reference to a CaseFileDefition to enforce certain schemata, and several additional meta-data attributes.

The CMMN specification defines decorators assignable to a plan item or discretionary item during the design phase. Table 2.1 provides a compatible matrix indicating which plan item or discretionary item supports which decorators. All items are notated generically and are representatives for specific items, i.e., the task item might be a human blocking task or any other task. In the following, each decorator is briefly explained:

- The Planning Table decorator indicates that a CasePlanModel, a Stage, or a HumanTask contain a discretionary item for run-time planning. The planning table decorator is graphically notated centered at the items top border.
- The Entry Criterion decorator indicates that a Stage, a Task, or a Milestone has specific criteria to enter this item. The entry criterion decorator is graphically notated centered at the item's border.
- The Exit Criterion decorator indicates that a CasePlanModel, Stage, or a Task has specific criteria that automatically exit the item. The exit criterion decorator is graphically notated centered at the item's border.
- The AutoComplete decorator indicates that a CasePlanModel or a Stage is completed automatically when no more actions are possible. The auto-complete decorator is graphically notated at the item's bottom.

- The `Manual Activation` decorator indicates that a `Task` or `Stage` requires a manual activation during run-time by a case-worker before execution. This decorator is typically used for a task that is used infrequently. The manual activation decorator is graphically notated at the item's bottom.
- The `Required` decorator indicates that a `Stage`, `Task`, or `Milestone` is needed to successfully accomplish a case. Nested elements within a stage that are not required itself are not affected. The required decorator is graphically notated at the item's bottom.
- The `Repetition` decorator indicates possible repetitions and is applicable for a `Stage`, a `Task`, or a `Milestone`. If a stage is repeated, all nested items are repeated similarly. The repetition decorator is graphically notated at the item's bottom.



Decorator Applicability	Planning Table ⊞	Entry Critrion ◇	Exit Criterion ◆	AutoComplete ■	Manual Activation ▷	Required !	Repetition #
CasePlanModel	☑		☑	☑			
Stage	☑	☑	☑	☑	☑	☑	☑
Task	HumanTask only	☑	☑		☑	☑	☑
MileStone		☑				☑	☑
EventListener							
CaseFileItem							
PlanFragment							

Table 2.1.: CMMN decorators applicability summary (Object Management Group, 2016).

Previously, relevant CMMN elements, related decorators and their applicability have been declared. Dependencies between items are expressed with `Sentries`. The sentry in Figure 2.7 expresses that the completion of `Task A` satisfies the `entity criteria` and enables `Task B`. The line is expressing dependency between items and is notated as a dash-dotted line.

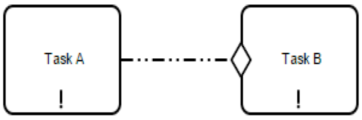


Figure 2.7.: Sentry-based dependency notated in CMMN (Object Management Group, 2016).

Figure 2.8 illustrates a more comprehensive example, expressing logical dependencies with sentries. A sentry is evaluated as satisfied when the linked items are completed. The left part shows a logical `AND` dependency, where `Task A` and `Task B` need to be completed so that `Task C` is enabled. The right part shows a logical `OR` dependency, where its sufficient when `Task A` or `Task B` is completed for `Task C` to be enabled. Additionally, the sample illustrates the required decorator.

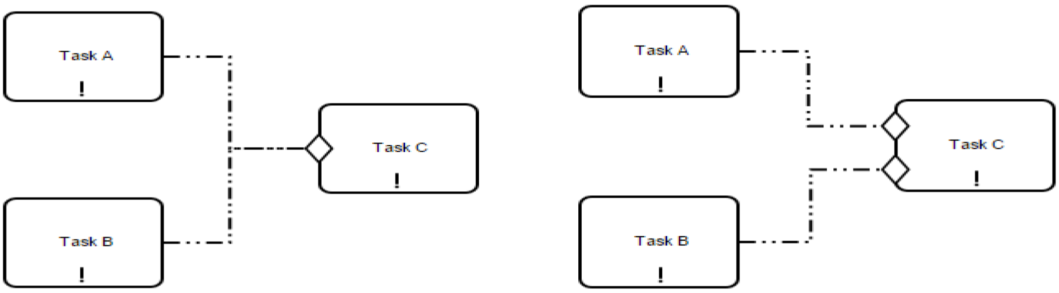


Figure 2.8.: Example sentries expressing a logical AND and OR dependency notated in CMMN (Object Management Group, 2016).

2.1.3. Classification of Existing Tools

Within the integrated care domain, there is mostly consensus about applying the ACM methodology for process-centric applications. In the integrated care context, the traditional BPMN approach has multiple shortcomings regarding flexibility and exception handling (Cano et al., 2015). Patient-centric treatments require teamwork which includes a significant amount of communication and coordination (Fischer, 2011, p. 183). Derived from the challenges presented in the problem description (cf. Section 1.1), three high-level capabilities required for integrated care have been identified, namely supported data-modeling, adaptive process modeling, and communication and coordination.

This classifications of existing tools section originally appeared in Michel et al. (2018) and is presented in the following with slight adaptations. We analyzed existing tools that could be used or combined to provide an integrated care tool support. New digital tools and new functionalities are continuously appearing while this analysis considers available information until May 2019. We selected an essential set of promising, well-known, and state-of-the-art tools based on the expertise and discussions between researches of an integrated care project. For the tool analysis, we used publicly available information such as documentation and tutorials which we extended with knowledge gained by attending vendor workshops. For the sake of completeness, it is important to mention that many analyzed tools might be programmatically extended to address required integrated care capabilities. We categorize the tools based on the typical problems they claim to solve. Therefore, we classified the tools into capabilities which are required to provide a collaborative, purely meta-model-based integrated care solution. Figure 2.9 presents a visual classification summary of the existing tools that are grouped according to the following categories:[2]

Data-centric approaches are based on the document management approaches and traditional social software (Matthes et al., 2011). In these cases, the process must be mapped to a basic operation (i.e., *Create*, *Read*, *Update*, and *Delete*) over the data. Therefore, the modeling process starts with the definition of the content model (e.g., the content types, fields, and their relations). The content model is fixed in cases such as *Wordpress*, *Typo3*, and *Confluence*, as a result, the process can be only mapped to the content workflow. However, other tools such as *Drupal*, *MediaWiki*, and *Trica* are based on explicit dynamic types so that the process can be mapped to the changes in the content and content types workflow. With the development of mobile technologies, a new set of tools has emerged based on the dynamic content to support integration with different formats of content (e.g., mobile notifications in *Parse* and *Firebase*). The flexibility to create, maintain, and evolve data and models in these tools, which can be extended using a user interface and role-based access rights models, and the support of social features and mobile devices makes these tools valid candidates to implement in the clinical case study. However, the main limitation of these tools is the lack of mechanisms to incorporate steps to the process that are not related to the data, which are necessary to document and reuse the process.

[2] The previous paragraph and the following paragraphs (Data-centric approaches, process-centric approaches, communication and coordination centric approaches, as well as the final paragraph) of the section 2.1.3 have already been published slightly different in Michel et al. (2018).

Process-centric approaches are further distinguished into ACM-based approaches and ad-hoc task-centric approaches. ACM is classically used for knowledge-intensive use cases where the degree of automation is low (Swenson, 2010). *Camunda* is an open-source workflow engine for large-scale processes that supports BPM, ACM, and decision modeling. Compared to similar tools, the implementation is conceptually close to the Case Management Model and Notation (CMMN) standard specified by the Object Management Group (2016). Within the ecosystem of *Camunda*, micro-services such as a CMMN based modeler, a generic case interface which might be a bit overwhelming for non-technical users, and a visual case tracing tool which helps to detect eventually occurring bottlenecks during case execution exist that can be assembled individually to an application landscape. An impressive feature is nesting an ACM subprocess into a superordinate BPMN process. *Ground Lion* is a sophisticated Belgian enterprise ACM tool provider that also provides a generic case client comparable with *Camunda*. The license fees for a minimal test setup, even for academic use are comparably high. *Camunda* and *Ground Lion* offer the possibility to model simple forms for a custom user interface representation. Other representative examples are *PEGA* and *IBM Case Manager*. *PEGA* is a platform for process automation that supports BPM and case management mainly for customer relationships. The *PEGA* platforms emerged from the CRM and marketing fields to reuse essential elements to engage customers. Additionally, it provides smooth integration with several enterprise areas such as marketing, sales, customer services, and similar areas. Complementary, *IBM Case Manager* provides complete generic case management capabilities. However, the separation between the user interface models, the process models, and data models requires coding skills or specific languages knowledge (e.g., layout components to align input fields). None of those tools supports extending the generic user interface representation without coding with domain-specific representations, such as a threshold-based coloring for numeric values, or a body visualization that indicates the patient's health status. All tools support a significant part of the full-stack meta-model-based approach, but mainly lack in supporting the required coordination and communication. Ad-hoc task-centric approaches are used for unstructured, not repeating use cases which might be comparable with certain CMMN run-time planning scenarios. *Trello* uses the Kanban principle to organize the ad-hoc tasks, whereas *todoist* is using a list-based approach. All three tools support granting access rights to collaborating members, in order to accomplish tasks collaboratively. Delegating tasks is also possible. The task state handling is solved in different ways, e.g., *Trello* changes the state by moving a task into the archive where both other tools allow to check a task. Conceptually, both tools are close to each other, all support indicating needed contribution and they support the integration of external data sources via REST APIs. Simple access rights are supported, but they are not role-based. *Trello* provides simple notifications created based on task models such as a task being overdue, but does not support the integration of custom domain-specific notifications. In addition, all three tools are only available as SaaS which is critical from the legal perspective.

Communication and Coordination Centric approaches are traditional email applications which provide a rich set of communication and coordination capabilities (Ducheneaut and Bellotti, 2001). Email applications emerged from traditional purely text-based messaging to contextual integrated communication and coordination applications supporting calendar management, contact organization, and simple task management to track deadlines. *Out-*

> *look* and *Thunderbird* are traditional email clients whereas *Gmail* provides a SaaS solution. All three support sending rich text messages to one or multiple persons, allow receiving messages, marking them as read, and they provide some tracking mechanisms. There are slight differences regarding the overall capabilities and their expressiveness. However, the most valuable capabilities regarding integrated care would be provided with tight integration into patient cases, while only comparably simple messaging is required. Contextual information is most valuable. Therefore, it would be desirable that alerts and notifications are directly assigned to process elements.

A dotted gray line indicates a collaborative purely meta-model-based ACM4IC approach classified according to the presented three dimensions in Figure 2.9. To the best of our knowledge, there is currently no purely model-based approach fitting those capabilities. Possible scenarios are to extend an available data-driven tool to support process and communication capabilities, to extend existing workflow engines with data-modeling and communication capabilities, or to use a generic workflow engine that is embedded in a hard-wired use case-specific application. The currently best practices were presented during a *Camunda* workshop in Munich on November 2016 and the most common examples and solutions that scale to several large-scale enterprises are the elaborated ones based on scenario three. However, the main limitation is the needed effort in building a use case-specific application. As a thumb rule, it was mentioned that on average approx. 10,000 case instances are required for the effort to pay off. Traditionally, customization is needed for each hospital site to comply with the local working culture. In the following, we will focus on scenario one, extending a meta-model-based approach to support the described high-level capabilities.

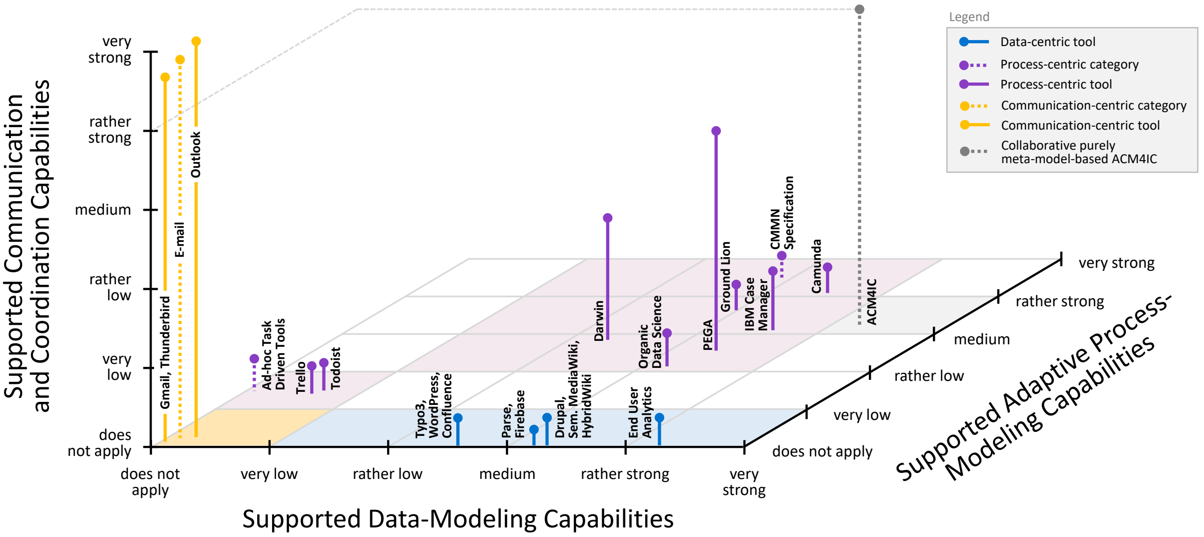


Figure 2.9.: A classification of existing tools regarding capabilities required for integrated care where conceptually, a collaborative purely meta-model-based approach is indicated in gray.

2.2. Related Work

First, a chronological overview elaborates the overall context in Section 2.2.1 and second, crucial highlights are depicted and explained in more detail in Section 2.2.2ff.

2.2.1. Chronological Overview

This section highlights related work chronologically ordered, heavily adapted and extended from Hauder (2016), as illustrated in Figure 2.10. Van der Aalst and Berens (2001) states that restrictive workflow approaches are used as a building blocks strategy to implement enterprise information systems whereas in a practical context, those often fail due to limited flexibility. Therefore, they argue that workflows should not be driven by the predefined control flows, but rather by the artifact to be created as presented with the case handling approach. Corresponding to that, Swenson (2010) described how adaptive case management influences the working methodology of knowledge workers. Independently, Büchner (2007) presents an approach for introspective model-driven software development. Those technical concepts enabled Matthes et al. (2011) to present a concept named Hybrid Wikis that allows business users collaboratively structure content with wiki pages. The first version of the CMMN specification published by the Object Management Group (2014b) enables declaring adaptive case models. Michel et al. (2015b) presents a collaborative approach named Organic Data Science (ODS) to decompose complex scientific work collaboratively into accomplishable tasks. The evolution from case handling to ACM is described by Marin et al. (2016). Hauder (2016) introduces Darwin, an approach that empowers end-users to collaboratively structure knowledge intensive-processes. Conceptually, Darwin applies collaboration concepts used for the ODS approach and combines those with template-based aspects from the Hybrid Wiki approach. In the meantime, the Object Management Group (2016) published the currently latest version of the CMMN specification version 1.1. Based on the Hybrid Wiki approach, the need for end-user analytics was identified, and a conceptual approach and prototypical implementation are presented by Reschenhofer and Matthes (2016a). Finally, previously presented concepts are combined and extended to enable a collaborative purely meta-model-based ACM4IC approach as elaborated by Michel and Matthes (2018) and Michel et al. (2018). The following subsections present most relevant concepts in more detail.

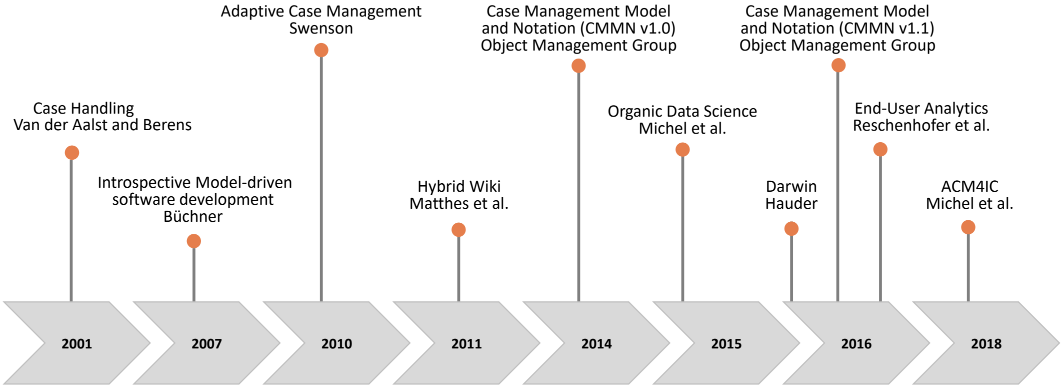


Figure 2.10.: Chronological overview adapted and extended from Hauder (2016).

2.2.2. Hybrid Wikis

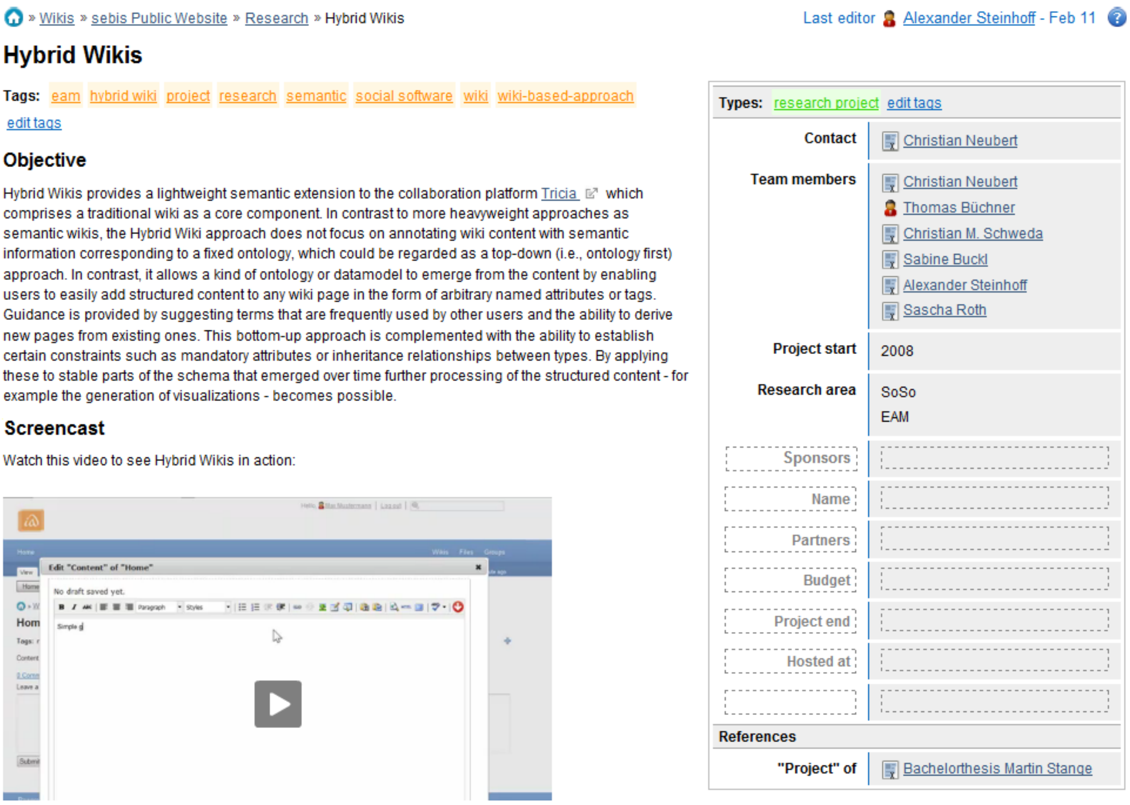
This section summarizes the original Hybrid Wiki approach presented by Matthes et al. (2011) and the dissertation published by Neubert (2012). An updated version of the Hybrid Wiki meta-model is presented by Reschenhofer et al. (2016). Typical wiki pages merely consist of text content, thus preventing information processing. The semantic wiki approach conceptually overcomes those issues by capturing structured information and enables declaring complex queries. However, business users are mostly not familiar with semantic annotations or their benefits, and often have difficulties entering them. The Hybrid Wiki approach provides familiar user interface elements for business users that are primarily a WYSIWYG editor for editing unstructured text content, a form-based input for entering structured information, and a spreadsheet-like interface for query results. Figure 2.11 illustrates the Hybrid Wiki approach; the unstructured information is presented on the left and the structured information is presented as a table sidebar on the right. Figure 2.12 represents the related meta-model indicating the most relevant conceptual capabilities which are introduced following:

Wiki The meta-model supports declaring multiple `Wikis` to enable encapsulated collaboration. A `Wiki` declares a space for collaboration between several stakeholders and supports declaring access rights such as an administrator, readers, contributors, and writers. The special contributor role allows creating new `WikiPages` and automatically grants write access to those, whereas existing pages are not accessible.

WikiPage A `WikiPage` consists of unstructured and structured content as illustrated in Figure 2.11. Structured content is represented with key-value pairs named `Attributes`. The well-known content tagging concept is extended with `TypeTags` that predefine the wiki page's structured content with `Attributes`. Multiple `AttributeDefinitions` are associated with a `TypeTagDefinition` that are applied when a certain `TypeTag` is assigned to a `WikiPage`.

Attribute All `Attributes` are either individually created for a specific wiki page instance or defined based on the associated `TypeTag` and related `TypeTagDefinition`. The loose coupling between the `Attribute` and `AttributeDefinition` enables a data-first and schema-second approach. An `Attribute` might have several `Values` represented with a `StringValue`, `DateValue`, `BooleanValue` or `LinkedValue`. A `LinkValue` enables referencing users, groups, or `WikiPages` to create a semantic ontology. `AttributeDefinitions` allow declaring `Validators` to ensure a certain `Attribute` multiplicity. When entering new `Values` on the user interface, the declared multiplicity is ensured, whereas when adapting an `MultiplicityValidator`, existing inconsistencies are allowed and highlighted to allow late schema modeling.

Based on the structured data provided with `Attributes`, the Hybrid Wiki approach allows declaring embedded queries to automatically generate views with the resulting `WikiPages` matching the query criteria. All queries consider the specified access rights and return user-specific results. The Hybrid Wiki approach is highly flexible and provides comprehensive data structuring capabilities including a data-first and schema-second strategy. However, task-centric aspects are not considered.



» Wikis » sebis Public Website » Research » Hybrid Wikis

Last editor Alexander Steinhoff - Feb 11

Hybrid Wikis

Tags: eam hybrid wiki project research semantic social software wiki wiki-based-approach
edit tags

Objective

Hybrid Wikis provides a lightweight semantic extension to the collaboration platform Tricia which comprises a traditional wiki as a core component. In contrast to more heavyweight approaches as semantic wikis, the Hybrid Wiki approach does not focus on annotating wiki content with semantic information corresponding to a fixed ontology, which could be regarded as a top-down (i.e., ontology first) approach. In contrast, it allows a kind of ontology or datamodel to emerge from the content by enabling users to easily add structured content to any wiki page in the form of arbitrary named attributes or tags. Guidance is provided by suggesting terms that are frequently used by other users and the ability to derive new pages from existing ones. This bottom-up approach is complemented with the ability to establish certain constraints such as mandatory attributes or inheritance relationships between types. By applying these to stable parts of the schema that emerged over time further processing of the structured content - for example the generation of visualizations - becomes possible.

Screencast

Watch this video to see Hybrid Wikis in action:

Types: research project edit tags	
Contact	Christian Neubert
Team members	Christian Neubert Thomas Büchner Christian M. Schweda Sabine Buckl Alexander Steinhoff Sascha Roth
Project start	2008
Research area	SoSo EAM
Sponsors	
Name	
Partners	
Budget	
Project end	
Hosted at	
References	
"Project" of	Bachelorthesis Martin Stange

Figure 2.11.: Hybrid Wiki page (Matthes et al., 2011).

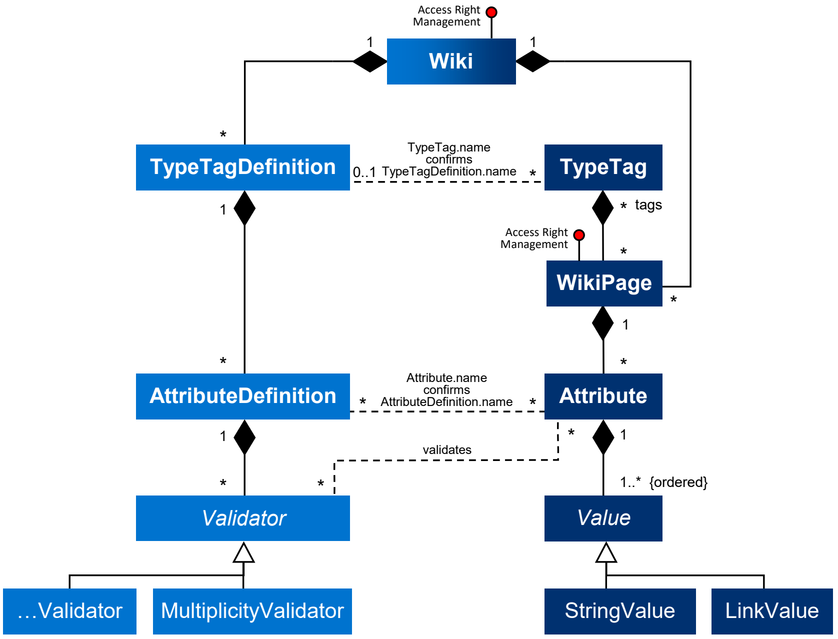
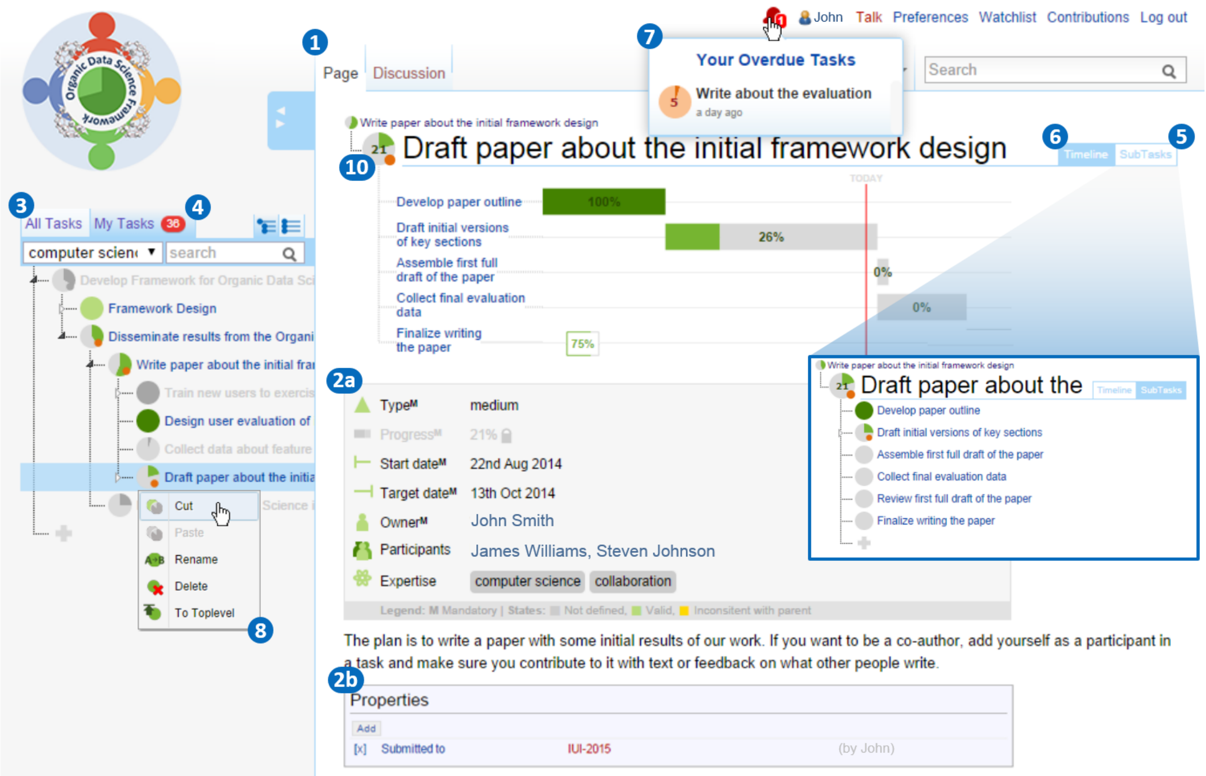


Figure 2.12.: Original Hybrid Wiki meta-model adapted from Matthes et al. (2011). An updated version of the Hybrid Wiki meta-model is presented by Reschenhofer et al. (2016).

2.2.3. Organic Data Science

This section summarizes the Organic Data Science approach from multiple publications (Gil et al., 2015b,c,a; Michel et al., 2015a,b; Michel, 2014). Scientific collaborations vary from closely collaborating work to loosely coordinated work (Bos et al., 2007). A few scientific collaborations share instruments (e.g., LHC at CERN), others share information from a joint database (e.g., Sloan Sky Digital Survey), share software (e.g., SciPy), or share a scientific question (e.g., Human Genome Project). The Organic Data Science approach supports scientific collaborations sharing a scientific question that need multi-disciplinary contributions, significant coordination, and the option to engage unanticipated researchers. Collaborations around scientific questions traditionally last a long time. (Gil et al., 2015b)

The Organic Data Science approach aims to support those collaborations with a space to decompose scientific questions into manageable tasks. Tasks are discussed, elaborated, and might be decomposed further if needed. Results gained from a task are aggregated to answer the decomposed scientific question. The design is grounded on best practices from Polymath, lessons learned from the ENCODE project, and social design principles that are derived from successful online communities (Michel et al., 2015b). Technically, the ODS design is implemented as a Semantic MediaWiki extension. Figure 2.13 highlights the most relevant conceptual features that are briefly described in Figure 2.14. On the left, a hierarchical task representation enables navigating within the decomposed scientific question. Similarly, a personal worklist provides filtered results corresponding to the actual user. On the right, the selected task is represented in detail, showing the parent task, the current task state with title, subtasks visualized by default as a Gantt chart, task meta-data, unstructured task content and structured task content as semantically linked data. The task meta-data block includes a task type, the progress, an expected start date, an expected end date, a responsible owner, participants who contribute, and an expertise field which allows assigning the expected required skills. All meta-data fields use the underlying semantic linked data structures to enable automatic task processing. Based on these properties, the aggregated progress for parent tasks is calculated automatically. The personalized worklist is dynamically generated based on the assigned owner and participants. User expertise's are automatically computed based on the required skills of accomplished tasks.


1
2a
2b
3
4
5
6
7
8
10
Organic Data Science Framework
All Tasks
My Tasks 36
computer scienc
search
Develop Framework for Organic Data Sci
Framework Design
Disseminate results from the Organi
Write paper about the initial fra
Train new users to exercis
Design user evaluation of
Collect data about feature
Draft paper about the initia
Science i
Cut
Paste
Rename
Delete
To Toplevel
John
Talk
Preferences
Watchlist
Contributions
Log out
Page
Discussion
Search
Timeline
SubTasks
TODAY
100%
26%
0%
0%
75%

Your Overdue Tasks

Write about the evaluation
a day ago

Write paper about the initial framework design

21 Draft paper about the initial framework design

Develop paper outline

Draft initial versions of key sections

Assemble first full draft of the paper

Collect final evaluation data

Finalize writing the paper

Type^M	medium
Progress^M	21%
Start date^M	22nd Aug 2014
Target date^M	13th Oct 2014
Owner^M	John Smith
Participants	James Williams, Steven Johnson
Expertise	computer science collaboration

Legend: M Mandatory | States: ▨ Not defined, 🟩 Valid, 🟨 Inconsitent with parent

The plan is to write a paper with some initial results of our work. If you want to be a co-author, add yourself as a participant in a task and make sure you contribute to it with text or feedback on what other people write.

Properties

Add

[x] Submitted to IUI-2015 (by John)

Write paper about the initial framework design

21 Draft paper about the

Develop paper outline

Draft initial versions of key sections

Assemble first full draft of the paper

Collect final evaluation data

Review first full draft of the paper

Finalize writing the paper

Figure 2.13.: Organic Data Science task-page highlighting most relevant conceptual features (Michel et al., 2015b).

⓿ **Welcome Page:** Describes clearly the science and technical project objectives, summarizes currently active tasks, and shows lead contributions (not shown).

❶ **Task Represntation:** Tasks have a unique identifier (URL), and are organized in a hierarchical subtask decomposition structure.

❷ **Task Metadata:** a) Describes clearly the science and technical project objectives, summarizes currently active tasks, and shows lead contributions. We distinguish between required metadata that is needed to progress a task and optional metadata. b) Optional user structured properties.

❸ **Task Navigation:** Tasks can expand until a leaf task is reached. Additionally users can search for task titles and apply an expertise filter.

❹ **Personal Worklis:t** The worklist contains the subset of tasks from the task navigation for which the user is owner or a participant. A red counter indicates the current number of tasks in the users worklist.

❺ **Subtask Navigation:** Subtasks of the currently opened task are presented. Filter and search options are not provided in this navigation.

❻ **Timeline Navigation:** All subtasks are represented based on their start, target times, and completion status in a visualization based on a Gantt chart.

❼ **Task Alert:** Signals when a task is not completed and the target date passed. A red counter next to the alert bell indicate the number of overdue tasks.

❽ **Task Management:** The interface supports creating, renaming, moving and deleting tasks. For usability reasons, all these actions can be reversed.

❾ **User Tasks and Expertise:** The interface allows users to easily see what others are working on or have done in the past. This creates a transparent process.

❿ **Task State:** Small icons visualize the state of each task intuitively. Tasks with incompleted required metadata are repesented with a cycle and tasks with completed required metadata are represented with a pie chart. The progess is indicated in green.

⓫ **Train New Members:** A separate site is used to train new users in a sandbox environment, where training tasks are explicit. The training is splited into two parts: 1) Users who participate on tasks and 2) User who own tasks (not shown).

Figure 2.14.: Organic Data Science conceptual features (Michel et al., 2015b).

Figure 2.15 shows a matrix of all possible task state visualizations on the left and sample task state transition sequences on the right. The right column representing active colored task state icons is applied for all tasks by default whereas the left column representing the fade-out task state icons is merely used to provide contextual information for non-matching tasks within a search. A task lifecycle is primarily separated into task states with incomplete meta-data, which are represented with a circle, where the colored sections indicate the percentage of the completed meta-data, and task states with completed meta-data, which are represented with a pie chart indicating the task content progress. Within the meta-data, a due date is specified when the task should be completed. To indicated needed contribution within nested task structures, overdue subtasks are highlighted with a small filled orange circle on all parent tasks. Overdue tasks themselves are highlighted with an orange colored pie chart representing the actual task progress. Completed tasks are indicated with a completely green circle. To support collaboratively decomposing a scientific challenge into manageable tasks, different actions aside from the decomposition are provided to refine the task's structure, such as moving a task, including all child tasks to another parent task. Considering that the moved task's due date is not compliant with the new parent task, inconsistent task state visualizations are introduced. Similar to the Hybrid Wiki validators, inconsistent task states are allowed when restructuring and prohibited for new entries to provide a maximum degree of flexibility. Inconsistent child tasks are indicated with a yellow triangle on the left, whereas an inconsistent task itself is represented with a yellow pie chart. The task state visualization primarily helps to quickly identify needed contribution. Inconsistency or overdue indications on parent tasks allow a fast hierarchical drill down.

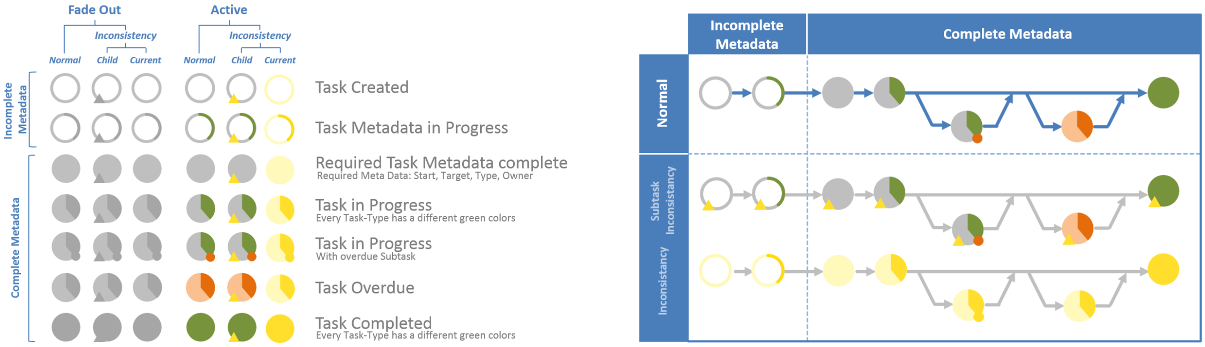


Figure 2.15.: Organic Data Science task states and sample transition sequences (Gil et al., 2015c).

The ODS is a comprehensive approach to ad-hoc decompose scientific challenges into manageable tasks which are primarily applicable for knowledge-intensive processes that are not predictable at all. However, templates that allow reusing pre-defined fragments within different contexts are not supported by the concept.

2.2.4. Darwin

This section summarizes the Darwin approach presented by Hauder (2016). Existing workflow management solutions are mostly restrictive and not suitable to typical knowledge-intensive processes which require dealing with uncertainties (van der Aalst and Berens, 2001). Within knowledge-intensive processes, the number of naturally occurring exceptions is high and not reasonably manageable with traditional process management systems (Strong and Miller, 1995). The applicability of adaptive case management in the context of enterprise architecture management is examined and related requirements are derived (Hauder et al., 2014). Darwin aims to support users to collaboratively structure knowledge-intensive processes. Developed and applied organic data science concepts such as social design principles (Michel et al., 2015b; Gil et al., 2015b; Michel et al., 2015a) and visual task information representation (Gil et al., 2015c,a) are incorporated into the Darwin approach. Figure 2.16 represents the conceptual user interface features and illustrates a wiki page with the corresponding subtasks.

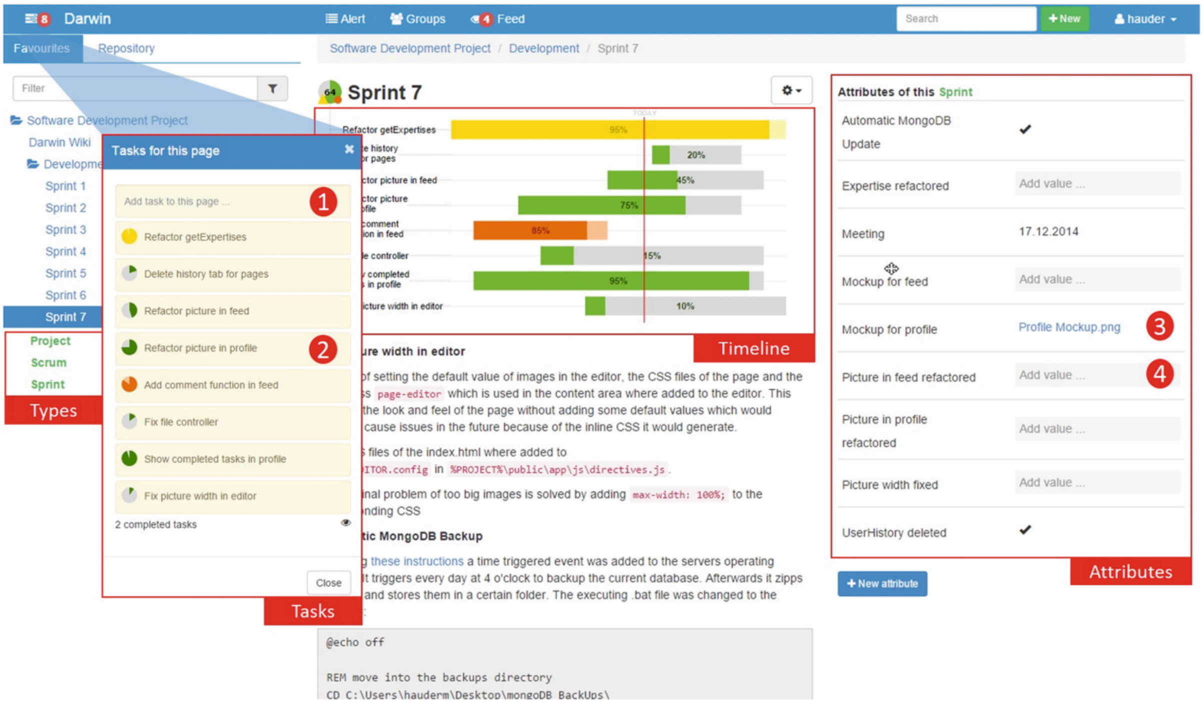
Conceptually, Darwin uses the Hybrid Wiki meta-model, which was extended with capabilities to manage knowledge-intensive processes. Technically, the Darwin system is built with a Scala-based framework independent from the Hybrid Wiki code base. Figure 2.17 pretenses the meta-model with a comparable abstraction as used for the Hybrid Wiki aggregated from multiple class diagrams (Hauder, 2016, p. 66ff.). The data schema and instance layer, colored in light blue and dark blue respectively, are conceptually comparable with the Hybrid Wiki, while access rights on the `Attribute` level are supported and the concepts are slightly renamed. The meta-model extensions to support templates for knowledge-intensive processes are colored in purple, related instances are colored in yellow and described in the following with more detail:

TaskDefinition Declaring a `TaskDefinition` empowers end-users to structure a workplan at run-time, similar to the emerging Hybrid Wiki approach enabling a data-first and schema-second strategy to provide a sufficient degree of flexibility. Conceptually, a `TaskDefinition` groups multiple `AttributeDefinitions` to a task template. All `Task` instances having an associated `TaskDefinition` show the `TaskDefinition` name.

Stage A `Stage` primarily describes the lifecycle within a `WikiPage`. Multiple `TaskDefinitions` may be associated with a `Stage` grouping related `TaskDefinitions`. Conceptually, organizing a workplan with `Stages` simplifies dependency modeling with `Rules`.

Rule Dependencies between `Process` elements respectively `Stages` and `Tasks` are modeled with `Rules` to express, i.e., a particular `Task` execution order. `Rules` are only applicable on model elements declared within a template which are `Stage` and `TaskDefinition`.

Task A `Task` is associated with exactly one `WikiPage` groups the `WikiPage Attributes` into accomplishable units. Each task supports meta-data fields such as a start date, end date, responsible users, and calculates automatically the process based on the number of accomplished attributes. Nested process structures are represented with a `LinkedValue` referencing the child `WikiPage` containing the child `Processes`. The automatic progress calculation considers the aggregated progress for the child `WikiPage` as progress for the `Attribute` referencing that `WikiPage`. Not applicable tasks in a particular context might actively be skipped by knowledge workers and further ignored for the progress calculation to enable accomplishing the overall process.



Tasks for this page

- Refactor getExpertises
- Delete history tab for pages
- Refactor picture in feed
- Refactor picture in profile
- Add comment function in feed
- Fix file controller
- Show completed tasks in profile
- Fix picture width in editor

2 completed tasks

ure width in editor

of setting the default value of images in the editor, the CSS files of the page and the
ss `page-editor` which is used in the content area where added to the editor. This
the look and feel of the page without adding some default values which would
cause issues in the future because of the inline CSS it would generate.

files of the index.html where added to
`ITOR.config` in `%PROJECT%\public\app\js\directives.js`.

inal problem of too big images is solved by adding `max-width: 100%;` to the
nding CSS

tic MongoDB Backup

g these instructions a time triggered event was added to the servers operating
It triggers every day at 4 o'clock to backup the current database. Afterwards it zipps
and stores them in a certain folder. The executing .bat file was changed to the

```
@echo off

REM move into the backups directory
CD C:\Users\hauderm\Desktop\mongoDB BackUps\
```

Attributes of this Sprint

Automatic MongoDB Update	✓
Expertise refactored	Add value ...
Meeting	17.12.2014
Mockup for feed	Add value ...
Mockup for profile	Profile Mockup.png
Picture in feed refactored	Add value ...
Picture in profile refactored	Add value ...
Picture width fixed	Add value ...
UserHistory deleted	✓

Figure 2.16.: Darwin task support for end-users (Hauder et al., 2015).

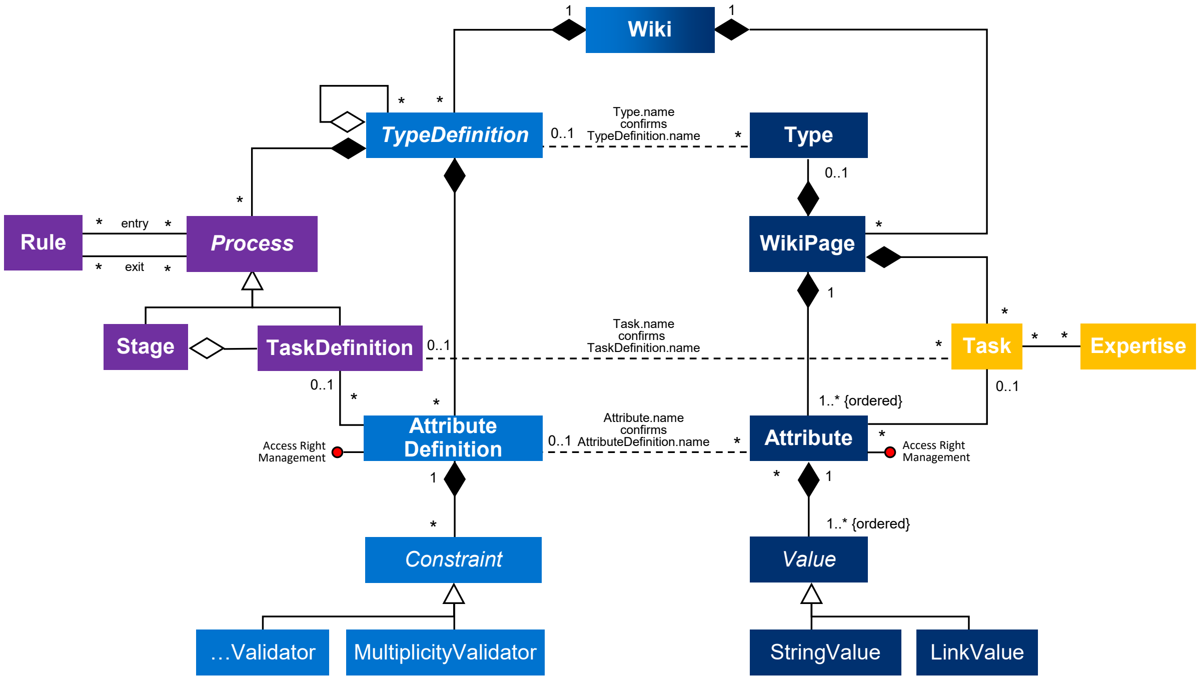


Figure 2.17.: Darwin meta-model aggregated from multiple class diagrams (Hauder, 2016, p. 66ff.) to be comparable with the Hybrid Wiki meta-model.

Darwin enables users to structure knowledge-intensive processes collectively. However, a seamless process orchestration across system boundaries is not supported. Furthermore, each model element is bound to precisely one default representation within the user interface which does not allow further customizations. Depending on the process size and the amount of collected data, a process summary might be helpful for knowledge workers.

2.2.5. End-User Analytics

This section summarizes the extension of the Hybrid Wiki meta-model with end-user analytical capabilities presented by Reschenhofer (2017). To realize the end-user analytical capabilities, a Model-Based Expression Language (MxL) was developed inspired through the Object Constraint Language (OCL) formerly developed by IBM. In recent years, OCL was enclosed to the UML standard specified by the Object Management Group (2014c, 2015). In the following, the crucial characteristics of MxL according to Reschenhofer (2013) are summarized: The *functional* paradigm which is reasonable for complex queries accessing complex data objects as stated by Buneman et al. (1995) and to prevent side effects. The Hybrid Wiki meta-model enables users to model complex linked objects, hence MxL is an *object-oriented* language. A query language supporting filter and transform operations for data sequences are considered as *sequence-oriented*. At compile-time, semantic consistency including user-defined model elements, such as referenced `EntityDefinitions` and `AttributeDefinitions` is ensured, hence MxL is *statistically type-safe*. The Hybrid Wiki approach supports versioning and the expression language supports accessing the past state of the data model. Therefore, MxL is also *temporal*.

Figure 2.18 illustrates the extended Hybrid Wiki meta-model to support end-user analytic capabilities with MxL. Primarily the concept of `DerivedAttributes` and `CustomFunctions` are incorporated, while the existing meta-model elements such as the `EntityType` and `AttributeDefinition` are extended with interfaces allowing to reference those elements. Newly introduced concepts provide an additional interface to define MxL expressions using the MxL type system illustrated in a simplified form on the left. While evaluating model-based expressions, the declared access rights are considered.

DerivedAttributeDefinition Compared to a traditional `AttributeDefinition` introduced within the Hybrid Wiki meta-model, a `DerivedAttributeDefinition` is computed automatically. Users are empowered to enrich model declarations directly with incorporated analytical capabilities expressible with MxL. Integrated models ensure a consistent declaration when model changes are applied.

CustomFunction Reusability aspects are addressed with the `CustomFunctions` that are either directly associated on a local `Workspace` or declared globally. A `CustomFunction` enables declaring computation rules, thus providing analytical abstraction which is may be reused within other `CustomFunction` declarations or `DerivedAttributeDefinitions`.

Figure 2.19 illustrates the visualizer application using MxL to calculate KPIs dynamically. Primary information sources usable to declare model-based expressions are entities, their attributes or derived attributes, whereas only the typed elements are supported. The visualizer supports creating custom dashboards depending on the domain-specific requirements.

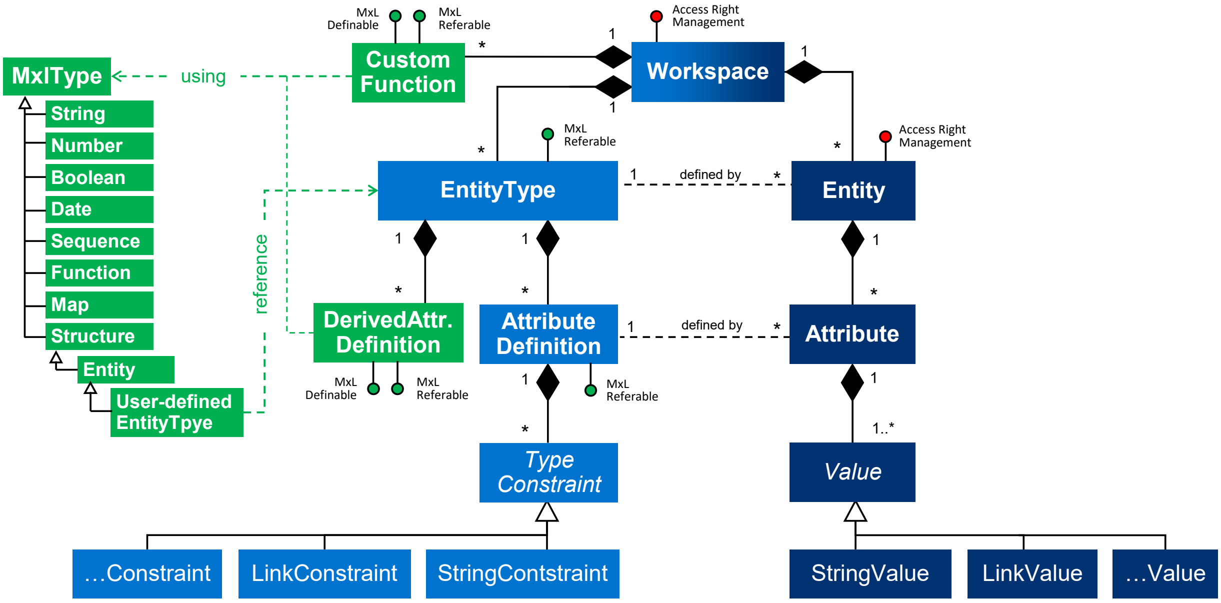


Figure 2.18.: End-user analytics meta-model with simplified MxL capabilities to be comparable with the original Hybrid Wiki meta-model (Reschenhofer and Matthes, 2016b; Reschenhofer, 2017).

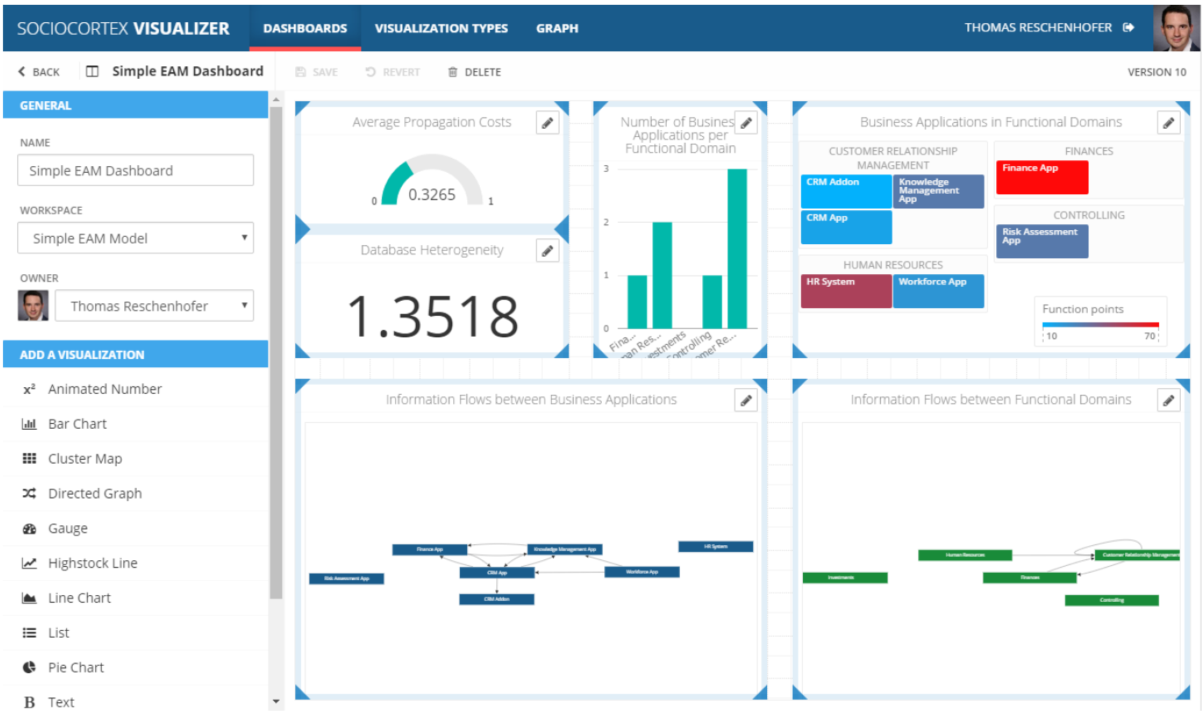


Figure 2.19.: End-user analytics visual interface showing KPIs calculated dynamically using MxL (Reschenhofer, 2017).

CHAPTER 3

Requirements

3.1. Requirements

This section presents the requirements derived from the literature. An essential requirement for software supporting adaptive case management is the ability to support all activities holistically required to accomplish a case (Burns, 2011). We identified three high-level requirements that are further decomposed into more specific requirements and elaborated in the following sections. In addition to the following explicitly elaborated functional requirements, specific non-functional requirements are obligatory. For instance, the system design must ensure the suitability as a Software as a Service (SaaS) solution. A SaaS approach also implicitly requires multi-tenancy support. Furthermore, a container-based deployment with Docker is preferred to ensure a reliable and fast deployment process (Bernstein, 2014). Lastly, for system interoperability and extendability, API interfaces must be provided.[1]

3.1.1. Support a Purely Meta-Model-Based Approach (R1)

A central requirement for an innovative adaptive case management approach is to replace the traditional system customization with a meta-model-based configuration. Supporting purely meta-model-based configurations across all system layers, such as data schemata, process models, discretionary access rights, and user interface models enables a flexible system setup to match desired needs consistently, rather than requiring an adaptation of the application code. (Matthias, 2011)

[1] The requirements detailed in sections 3.1.1, 3.1.2, and 3.1.3 have already been published slightly different in Michel and Matthes (2018).

R1.1 Support Data Schema Models

The data schema models must support modeling the data generated during the case execution. In addition, data resulting from third-party systems integration, such as patient data from hospital information systems must be modeled. (adapted from Michel and Matthes, 2018)

Within case management, data and processes are tightly integrated (Hauder et al., 2014). According to the CMMN specification, case related data is either directly stored within the case or references the primary information source. Data is encapsulated in referable objects to enable nested and reusable data structures. Each data object typically consists of multiple typed key-value pairs representing the actual information (Object Management Group, 2016). However, the CMMN specification is merely seen as a reference to support standards and applied where appropriate while allowing deviations where needed.

R1.2 Support Adaptive Process Models

The system must support defining adaptive treatment templates that are customizable to hospital and treatment specific requirements. As a reference methodology for defining the processes, Adaptive Case Management (ACM) should be used. To support integrated care, these processes need to be synchronizable with other systems. (adapted from Michel and Matthes, 2018)

Clinical processes vary from structured administrative processes to unpredictable patient treatment processes. A crucial characteristic of ACM is to support unpredictable situations, while it also comprises partly structuring concepts from the traditional workflow approaches (Burns, 2011). Processes are characterized based on the degree of structure (Di Ciccio et al., 2012; Michel et al., 2018). Figure 3.1 distinguishes between four process categories. Strongly structured processed that do not allow any exceptions during execution are typical BPMN processes. Structured possess with ah-hoc exceptions during execution and unstructured processes with predefined fragments are typical ACM processes. Completely unstructured processes are performed on an ad-hoc basis without a case template. With a rising knowledge intensity and higher flexibility, the automation intensity or predictability decreases, and vice versa. In clinical practice, all categories occur and must be supported. Considering the clinical environment with limited human resources and time pressure, it is unlikely that professionals structure unpredictable processes similarly as costly structured processes with key-value pairs. In practice, a set of predefined process fragments is applicable, or unstructured information is documented in a text-based form as current best-practice (cf. R3.3). Temporal dependencies between tasks and stages must be supported (Hauder et al., 2014; Kurz et al., 2015). According to the CMMN specification, those are expressed with a concept named sentries (Object Management Group, 2016). When multiple process elements must be accomplished, sentries must enable to express those dependencies with a logical *and*. Additionally, dependencies based on case data must be expressible. Run-time planning is crucial to handle unpredictable situations (Hauder et al., 2014; Kurz et al., 2015). While the CMMN specification declares a complex run-time planning concept, a simple concept should allow adding predefined stages and tasks dynamically during run-time.

Transparent responsibility expressing which case-worker is responsible for accomplishing which task until when help facilitate case-work and enable interventions when needed (Hauder et al., 2014). The conceptual design should use the CMMN specification where suitable.

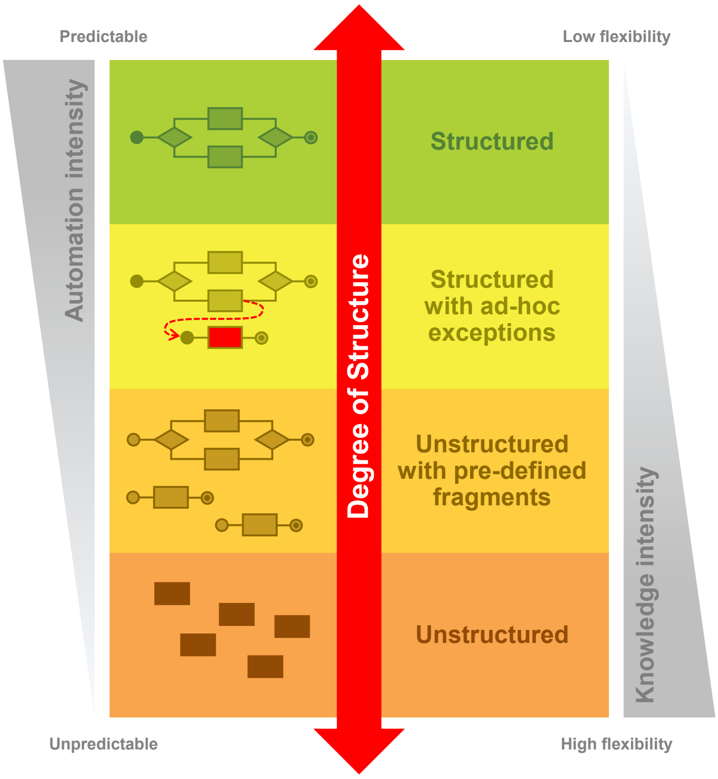


Figure 3.1.: Degree of structure according to Michel et al. (2018), originally adapted from Di Ciccio et al. (2012).

R1.3 Support Role-Based and Discretionary Access Right Models

The system needs to support granular role-based and discretionary access control mechanisms to define which care professionals are allowed to access which patient cases. In addition, clinical tasks must be assignable based on roles. (adapted from Michel and Matthes, 2018)

Integrated care requires accessing sensitive patient data while data privacy must be ensured. Typically, multiple care professionals with multiple roles work collaboratively on a case (White, 2009), whereas the level of involvement may differ (Burns, 2011). As knowledge work is characteristically not predictable upfront, it must be supported to grant access dynamically during run-time (Hauder et al., 2014; Kurz et al., 2015). A read, write, and owner access level for the case responsible care professional must be supported.

R1.4 Support Simple User Interface Models

In general, the user interface must represent each model element in a generic model-based manner while supporting simple layouting mechanisms for tasks. The default representation must be overridable with a declared special representation to support dedicated integrated care use cases and ensure extendability. (adapted from Michel and Matthes, 2018)

The user interface is a critical success factor to engage end-users. Tractinsky et al. detected a strong correlation between the awareness aesthetics and the awareness usability of a system (Tractinsky et al., 2000). The user interface represents the application, whereas the quality significantly influences the usability and acceptability (Suduc et al., 2010). Therefore, it is crucial to use default representation where possible, but to support deviating from default representations were required to minimize the effort and maximize the benefit. Table 3.1 summarizes the above elaborated first requirement, including its subordinated more specific requirements and references the sources from the literature.

ID	Requirement	Source
R1	Support a purely meta-model-based approach	Matthias 2011
R1.1	Support data schema models	Hauder et al. 2014, Object Management Group 2016
R1.2	Support adaptive process models	Burns 2011, Di Ciccio et al. 2012, Hauder et al. 2014, Kurz et al. 2015, Object Management Group 2016, Michel et al. 2018
R1.3	Support role-based access right models	Burns 2011, Kurz et al. 2015, Hauder et al. 2014, White 2009
R1.4	Support simple user interface models	Tractinsky et al. 2000, Suduc et al. 2010

Table 3.1.: Summary of requirement sources (R1).

3.1.2. Support Third-Party System Integration (R2)

Knowledge workers typically require information from multiple information sources, non-integrated information sources interrupt their work and delay complying activities (Matthias, 2010). Integrated care mainly relies on aggregated information to achieve a patient-centric treatment. Primarily, there are two design patterns available to design system interoperability, namely a process-based integration, a data-driven integration, or a combination of both (Kurz et al., 2015).

R2.1 Support External User Identity Management

The system needs to provide a Single Sign-On (SSO) for the authentication of care professionals. Therefore, an external user identity management must be supported. Foreign identifiers should be used as internal primary keys to simplify the integration. (adapted from Michel and Matthes, 2018)

Nowadays, the architectural micro-service pattern is widely used to apply the separation of concerns strategy. In the medical domain, most services require user authentication due to confidential patient data. A centralized user identity management enables a consistent authentication policy across service boundaries and enables a Single Single-On strategy (De Clercq, 2002). After the successful login, encrypted tokens are typically generated, handed over to the client, and passed with every request to the related micro-service (Bánáti et al., 2018). The case authorization mainly depends on the instantiated case and therefore, it should be managed within the case execution engine.

R2.2 Support Process Orchestration of Third-Party Systems
To provide an integrated care service, the system needs to support the orchestration of external systems. External system processes, such as patient self-management processes, should seamlessly be integrated to provide aggregated process information for care professionals. (adapted from Michel and Matthes, 2018)

Although ACM traditionally has a low degree of automation, external events are affecting the case execution (White, 2009). In integrated care, patient-triggered events, such as monitoring data that exceeds declared thresholds within an external monitoring system must be considered. Therefore, a holistic integrated process orchestration across system boundaries is required (Kurz et al., 2015).

R2.3 Support Semantic Integration of External Data Sources
Hospital information systems with extensive amounts of patient data exist. Therefore, system architecture needs to consider enhancing internal data with external data sources. (Michel and Matthes, 2018)

Complexity for case-workers is often driven by information that is distributed across multiple information systems (Matthias, 2010). A key characteristic of case management is the low degree of automation; however, data from some legacy information system is typically required (White, 2009). A holistic case management approach requires seamless integration of those existing information sources to provide future solutions (White, 2009). Typically, patient information is already managed within one or more hospital information systems that might be integrated to prevent entering existing data multiple times. Table 3.2 summarizes the above elaborated second requirement, including its subordinated more specific requirements and references the sources from the literature.

ID	Requirement	Source
R2	Support third-party system integration	Kurz et al. 2015
R2.1	Support external user identity management	De Clercq 2002, Bánáti et al. 2018
R2.2	Support process orchestration of third-party systems	White 2009, Kurz et al. 2015
R2.3	Support semantic integration of external data sources	White 2009, Matthias 2010

Table 3.2.: Summary of requirement sources (R2).

3.1.3. Support Communication and Coordination (R3)

Traditionally, a case represents patient treatment that requires communication and coordination efforts. Within integrated care, teams are typically spread across organizational boundaries. A case team mostly comprises specialists for different disciplines, each responsible for certain aspects, while results must be documented and possible treatments must be discussed and with colleagues finally coordinated (White, 2009). Active information exchange is crucial, because professionals depend on advice and knowledge exchange (White, 2009).

R3.1 Support Process Contextual Notifications
The care professionals can prescribe patients certain tasks via a third-party system, such as blood pressure measurements every morning. If the blood pressure exceeds an individually specified threshold, the system should notify the responsible care professional. Unforeseen technical situations should also lead to notifications of professional users. (adapted from Michel and Matthes, 2018)

Knowledge-intensive work must handle unpredictable situations where a manual exception handling may be required. Tight information integration is important for an efficient case management (Matthias, 2010), therefore, notifications must be linked directly to the related process element or task causing the notification. Merely the process-responsible professional and the case owner should be notified to avoid the unnecessary distraction of the case-workers.

R3.2 Support Direct Case-Based Communication
Case-based professional-to-professional messages need to be supported for information exchange. Therefore, all involved care professionals should be able to follow ongoing conversations. Direct communication with the patient needs to be supported in a separate conversation to provide integrated care. (adapted from Michel and Matthes, 2018)

Nowadays, knowledge workers typically use multiple applications while accomplishing work, which includes email-based applications as established communication channel (Motahari-Nezhad and Swenson, 2013). Integrating email-centric communication is described as integration capability (Matthias, 2010). The cases contain sensitive patient data. Therefore, authentication and case-based authorization require a tight integration, where as rich text messaging features are sufficient.

R3.3 Support Unstructured Case Notes
In addition to the semi-structured processes, the system needs to provide a wiki-based notes area where care professionals can collaboratively document unstructured information as well. Individual predefined templates should be provided depending on case definitions. (adapted from Michel and Matthes, 2018)

To efficiently organize all contextual information relevant for a case and to support easy access is crucial, while decoupled information may get lost or not be accessible when needed (McCauley, 2011). Additionally, text-based case notes are a well-known concept and widely accepted in the medical domain. An optional template declaration should provide additional system flexibility through customization for dedicated treatments.

R3.4 Support Case-Template Specific Summaries

The case template must support declaring summary sections to provide template-specific summaries that help care professionals to identify crucial case information representing the patient's status quickly.

Simple generic case progress metrics do not represent the actual state and case progress sufficiently. Mathematical metrics may express the percentage of currently accomplished tasks, but can not sufficiently consider tasks that may later dynamically added during run-time planning. Care professionals understand the implicit conceptual case progress which results from existing case data (White, 2009). Therefore, declaring case-specific summary sections must be supported to enable the representation of crucial case information to make the case progress as transparent as possible. The need for visualization of the actual case progress is as well described by Kurz et al. (2015).

R3.5 Clarify Needed Contribution

A single case contains many tasks that need to be accomplished by either a care professional or the patient. The system needs to support assigning tasks to care professionals based on their roles. Traditionally, care professionals have multiple cases. Therefore the system needs to provide a dashboard to indicate where contributions are needed. (adapted from Michel and Matthes, 2018)

Case-work is collaborative and knowledge-intensive, which implies a variety of individual decisions are required that must be documented. To ensure that the case-work is accomplished, needed contribution and responsibility must be transparent for all case-workers (Hauder et al., 2014; Kurz et al., 2015). When responsibilities are changing, the system must allow updating those to represent the actual responsibilities. Table 3.3 summarizes the above elaborated third requirement, including its subordinated more specific requirements and references the sources from the literature.

ID	Requirement	Source
R3	Communication and coordination	White 2009
R3.1	Support notifications	Matthias 2010
R3.2	Support communication within a case	Matthias 2010, Motahari-Nezhad and Swenson 2013
R3.3	Support case notes	McCauley 2011
R3.4	Support template specific summaries	White 2009, Kurz et al. 2015
R3.5	Clarify needed contribution	Hauder et al. 2014, Kurz et al. 2015

Table 3.3.: Summary of requirement sources (R3).

CHAPTER 4

Conceptual Design

This chapter focuses on the conceptual design which includes the conceptual architectural layers presented in Section 4.1, the conceptual meta-model as illustrated in Section 4.2, the execution semantics that is described in Section 4.3 and relevant design challenges with conceptual solutions in Section 4.4. Finally, supported modeling elements according to the CMMN notation are presented in Section 4.5, and supported requirements are summarized in Section 4.6.

4.1. Conceptual Layers

Our design consists of several conceptual abstraction layers that have initially been published by Hernandez-Mendez et al. (2018). This section presents the partly advanced conceptual layers with a focus on adaptive case management. Figure 4.1 illustrates the latest version. All persisted information is stored in a single relational database to allow linking elements of the different conceptual layers to create a large ontology. Publicly accessible functionality is exposed via a RESTful API that considers roles and access rights. Higher conceptual layers use the capabilities of the lower conceptual layers and provide higher abstracted functionality. Therefore, the conceptual layers are described in detail, beginning with the lowest layer, the annotated versioned linked content graph. Each conceptual layer is associated with a particular color. The uniform color scheme is used across the entire thesis to associate with the conceptual layers easily.[1]

Annotated Versioned Linked Content Graph The first layer supports versioned data storage and ensures the capability of the system to store revisions semi-structured data in a form of entities (i. e., JSON objects) and references between them. The rationale is that companies already manage structured and unstructured data, so it is necessary to create a mechanism for the system to import an initial set of semi-structured data (e. g., Excel sheets, wiki pages with embedded tables and media) without strong data schema that can evolve over the time. This layer supports the data-first (schema-second) approach for data modeling (Büchner, 2007). This paragraph originally appeared in Hernandez-Mendez et al. (2018).

[1] The following paragraphs regarding the conceptual layers in section 4.1 have already been published in Hernandez-Mendez et al. (2018). They have been adopted, slightly adjusted, or heavily adjusted (cf. remark at the end of each conceptual layer paragraph).

🟦 **Multiple Dynamic Schemata** This layer allows the users of the information system to collaboratively structure their information over time by adding or removing schema constraints involving relationship and cardinality constraints as needed. A data schema is thus not seen as a definition of a container that is subsequently filled with data (like an SQL database), but as a collection of constraints imposed on a flexible content graph that evolves over time. However, this implies that at certain times during system evolution the data can contain inconsistencies, which can be resolved collaboratively without leaving the scope of the system. This paragraph was summarized from Büchner (2007); Matthes et al. (2011) and originally appeared in Hernandez-Mendez et al. (2018).

🟥 **Role-Based and Discretionary Access Control Models** This layer addresses the security concerns in the system. The security model comprises users and groups which can be internal or external to the organization. The access rights (administrator, editor, author, and reader roles) are defined initially at the Workspace level and can be overwritten (loosened or tightened) at the Entity level. The rationale is to guarantee secure access to the data stored in the system. This paragraph originally appeared in Hernandez-Mendez et al. (2018).

🟧 **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 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 the fact that not all the data in the enterprise is structured or semi-structured and the system should be able to use all the possible formats of the data in the enterprise without the need of an upfront structuring process. This paragraph originally appeared in Hernandez-Mendez et al. (2018).

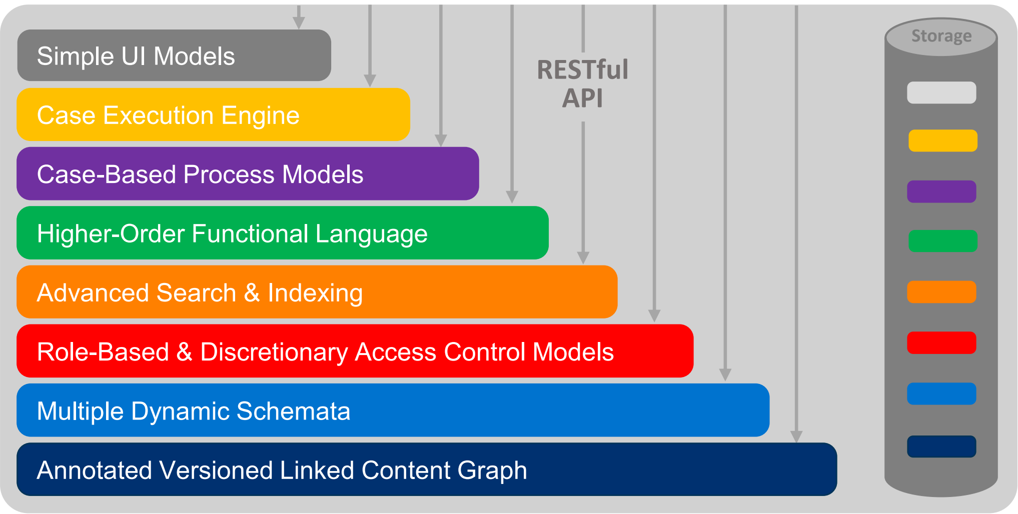

Simple UI Models
Case Execution Engine
Case-Based Process Models
Higher-Order Functional Language
Advanced Search & Indexing
Role-Based & Discretionary Access Control Models
Multiple Dynamic Schemata
Annotated Versioned Linked Content Graph
RESTful API
Storage

Figure 4.1.: Conceptual architectural layers adapted from Hernandez-Mendez et al. (2018).

🟩 **Higher-Order Functional Language** 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. This language allows adding new attributes to Entities as a result of the computations (i. e., Derived Attributes) and using operations over collections of Entities such as Map, Reduce, and Join. The rationale behind that is creating a flexible mechanism to the user of the system to extract knowledge from the stored data, which is not limited by the basic API of the system and allows the users to create flexible and tailored data views based on individual information demands. This paragraph was summarized from Reschenhofer and Matthes (2016a) and Reschenhofer (2017) and originally appeared in Hernandez-Mendez et al. (2018).

🟪 **Case-Based Process Models** This layer allows the user to define knowledge- and data-intensive processes following the adaptive case management paradigm introduced by Swenson (2010). The conceptual design uses the CMMN 1.1 (Object Management Group, 2016) specification as a reference and adapts those where needed. A modeler can define stages, tasks, and sentries to declare a collaborative process. Each task links to one or multiple attributes which are read-only or writable. Thereby, the design allows the modeler defining standard processes or reusable process fragments which can be dynamically instantiated as needed. Case templates, including required data structures, are declared as XML files which are imported for execution. This paragraph was slightly adapted from Hernandez-Mendez et al. (2018).

🟨 **Case Execution Engine** This layer manages the information regarding all the case instances being executed in the system and the links to the shared or case-local data and the actors involved in a case. The case execution engine enforces the access control policies defined in the access rights layer and the data management layer keeps an audit trail of the executed steps and data modifications. The rationale is to keep track of the process steps execution and how the users accomplished the case to use for future analyses as well (e.g., for compliance or prediction purposes). Additionally, communication and coordination features, such as case-based messaging, notes, and alerts are provided. This paragraph was slightly adapted from Hernandez-Mendez et al. (2018).

◼ **Simple User Interface Models** This layer allows to customize the data representation on the user interface. The schemata layer provides a number of basic data types, with each data type being bound to exactly one generic readable and writable representation. The default representation might be overridden if needed to provide a custom data representation. Additionally, simple grind layout options are provided. This paragraph focuses on the relevant features for ACM4IC and was heavily adapted from Hernandez-Mendez et al. (2018).

Figure 4.2 shows all the capabilities that are provided in each layer of the architecture and their semantic relationships (capability A uses capability B, capability A extends capability B). The color coding links the elements to the layers depicted in Figure 4.1 and the concepts in the conceptual model in Figure 4.3. We found this capability map to be a very useful starting point for core developers of the platform to understand the key dependencies in the architecture. This paragraph was slightly adapted from Hernandez-Mendez et al. (2018).

In the following, the thesis will primarily focus on the last three layers that extend the data-driven architecture to support adaptive case management. Lower level concepts might be presented with a higher abstraction degree to provide a holistic focus on the conceptual design.

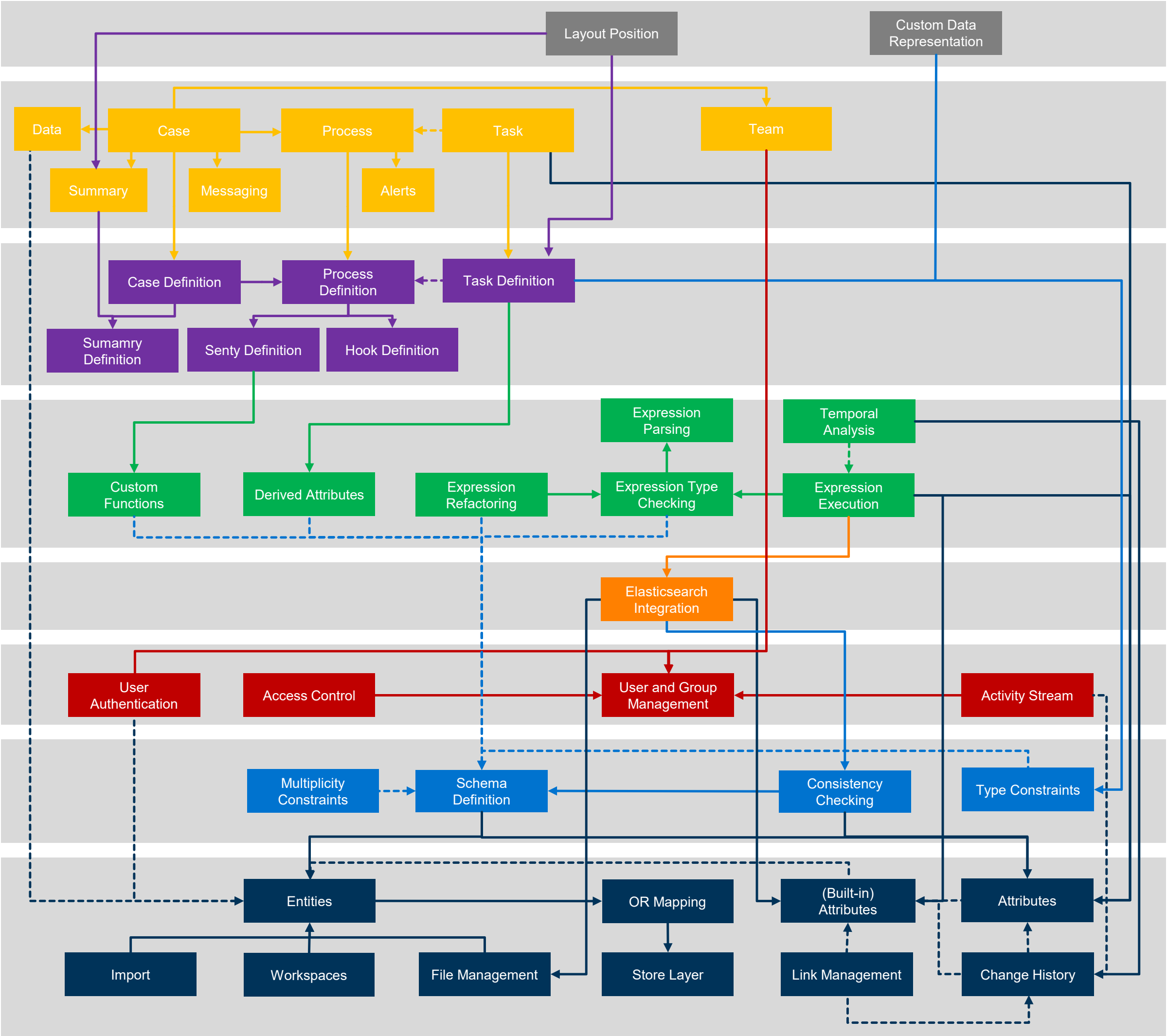


Figure 4.2.: Capabilities ordered according to conceptual layers. A solid line represents the usage of a particular capability and a dashed line represents extended functionality. Adapted from Hernandez-Mendez et al. (2018).

4.2. Meta-Model

The conceptual meta-model sketches modeling concepts within each conceptual layer and clarifies the central dependencies across the conceptual layers. We published an initial version of the conceptual meta-model in Hernandez-Mendez et al. (2018). Figure 4.3 illustrates the finalized conceptual meta-model, a gray box in the background indicates a specific layer, whereas all modeling elements are color-coded according to the related layer similar as in Section 4.1. The class diagram primarily focuses on relevant concepts for a holistic ACM design and simplifies related concepts where possible. In the following, the modeling elements are introduced and described in context.

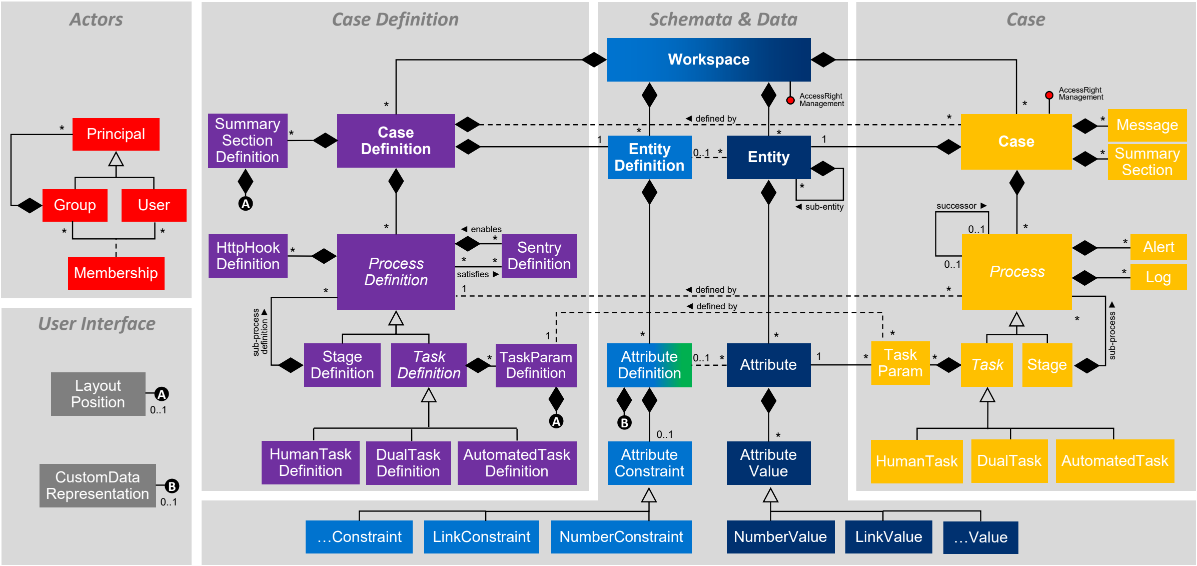


Figure 4.3.: Meta-model adapted from Hernandez-Mendez et al. (2018). A more detailed conceptual meta-model containing most relevant attributes is illustrated in Figure A.1.

A Workspace is a container for model definitions and their instances. Therefore the Workspace is simultaneously associated with the schemata and data layer. Additionally, access rights such as readers, writers, administrators, and contributors might be declared. The contributor role might instantiate model elements and get explicit write access for those. All access rights accept one or multiple Principals as value. By default, the access rights are inherited to the containing modeling elements with some exceptions. The inherited access rights might be extended with dedicated access rights for specific Cases. Typically, a Workspace defines a protected environment accessible for a specific stakeholder group which might represent a classical tenant.

The User and Group management is globally accessible to enable sharing Workspaces with multiple Groups. A composite pattern is applied where the Group represents the composite, the User represents the leaf, and the Principal is the abstract component. The expressive power allows nesting Groups. A Membership expresses that a User belongs to a specific Group and contains meta-data such as creation date and possibly an expire date. A Group Membership implicitly includes nested Groups.

An EntityDefinition in combination with AttributeDefinitions defines a data structure comparable with a class and their attributes in the UML notation (Object Management Group, 2015). Similarly, an Entity in combination with an Attribute represents instantiated data comparable with an object and its attribute values in the UML notation. The loosely coupled data and schemata layer enables late modeling with a data-first and schema-second strategy, whereas late modeling is not supported in combination with case management.

An AttributeDefinition is associated with exactly one EntityDefinition and has may an optional AttributeConstraint that allows typing an AttributeDefinition. Similarly, an Attribute is associated with exactly one Entity and supports multiple AttributeValues depending on the modeled multiplicity. Available multiplicity options are any, maximum one, exactly one, or at least one. The abstract `AttributeConstraint` enforces a specific `AttributeValue` on instance level. The `AttributeValues` are categorized into primitive types representing a string, a longtext, a number, a date, an enumeration, a JSON object and into complex types that represent a referenced value.

The `NumberConstraint` enforces a `NumberValue` and additionally supports declaring a minimum and a maximum `NumberValue`. The `EnumerationConstraint` allows declaring enumeration options that contain a value and a value related description. The separation between value and description allows using the value for calculations while having a human-readable textual description. The `LinkConstraint` enforces a `LinkValue` which links instantiated model elements, such as an `Entity` or a `Principal`, a `Group`, or a `User`. A `LinkConstraint` referring an `Entity` supports the optional declaration of an allowed `EntityDefinition`. Additionally, a `LinkConstraint` referring a `User` optionally supports declaring a set of `Groups` where the `User` must have a `Membership` in at least one of those.

Additionally, the meta-model enables defining `DerivedAttributeDefinitions` that are declared with expressions and calculates the related value, typically depending on other `AttributeDefinitions`. For reason of simplification, the `DerivedAttributeDefinitions` are indicated as part of the `AttributeDefinition` with a green background color. Conceptually, all instantiated `DerivedAttributes` are evaluated dynamically and therefore, not referable with a unique identifier.

A `CaseDefinition` bundles all model elements to collectively declare a holistic, purely meta-model-based `Case` template. Each `CaseDefinition` references to exactly one `EntityDefinition` to define the `Case` root data structure. Optionally, second `EntityDefinition` is referenced to extend the `Case` root `Entity` directly after the `Case` instantiation. Attaching this additional data structure allows to declare the `Case` roles and due dates encapsulated from the actual workflow data. Each `Case` has an owner who is responsible for managing the `Case`. Therefore, the `CaseDefinition` declares an owner path depending on the `Case` root entity. Similarly, a `Case` client is declared that represents the treated patient. During the `CaseDefinition` import, a version is declared, and for all older versions, the instantiable flag is automatically set to false by default. Optionally, a notes template can be declared.

A composite pattern enables defining nested workflows. The `StageDefinition` represents the composite and the abstract `TaskDefinition` with its inherited special task types represents the leaf. A `TaskDefinition` is either an abstract `TimedTaskDefinition` which supports declaring a due date path or an `AutomatedTaskDefinition`. The abstract `TimedTaskDefinition` represents either a `HumanTaskDefinition` or a `DualTaskDefinition`. A `HumanTask` is executed by a human, an `AutomatedTask` is typically executed by a third-party system, and a `DualTask` represents a `HumanTask` followed by an `AutomatedTask`. Within the composite pattern, the `ProcessDefinition` represents the abstract component.

The abstract `ProcessDefinition` forms the basis for declaring workflow crucial `Process` properties. By default, a `Process` element is repeatable once, or if declared, serial or parallel repeatable. A flag indicated whether a `Process` is mandatory or not. An automatic `Process` activation is performed by default, while alternatively, a manual activation or an expression which is evaluated to an automatic or manual activation during run-time can be declared. Each Process should have a responsible User who is in charge to accomplish the task and to handle related unforeseen events. Therefore, the ProcessDefinition allows declaring an owner path that describes a path depending from the Case root Entity, which is resolved during run-time. Typically, all Processes generate and extend the existing Case data. Therefore, a new entity attach path depending on the Case root Entity in combination with an EntityDefinition reference can be defined to extend the hierarchical Case Entity structure.

The abstract TaskDefinition typically has several TaskParamDefinitions that declare the tasks in an output. A TaskParamDefinition has properties that define if the parameter is readable or writable, if the parameter is mandatory, and declares a path that links to the data layer depending on the Case root Entity. This approach allows binding Attributes to TaskParams to define the Tasks data structure. Attributes may bound to multiple TaskParams to update previous results.

SentryDefinitions are declarable for all inherited ProcessDefinitions to model complex preconditions. A SentryDefinition has may multiple ProcessDefinitions that need to be completed to satisfy the SentryDefinition. Optionally, artifact-centric dependencies are declared with expressions on the data layer. These expressions are only evaluated after the ProcessDefinitions preconditions are satisfied.

To enable an integrated process orchestration across system boundaries, HttpHookDefinitions allow declaring hooks on Process state change events, such as activate, enable, complete, terminate, or a not state changing Task correct event. A URL and HTTP method, such as POST, GET, PUT, or DELETE must be specified. Optionally, a failure message can be declared. The CaseDefinition also allows declaring hooks on state change events and on a Case delete event. Compared to the ProcessDefinition hooks, the CaseDefinition hooks provide limited capabilities. Simply one hook can be declared per event, while only the HTTP GET method is supported and failure messages are not supported.

A CaseDefinition supports modeling multiple SummarySectionDefinitions that, when combined, provide a customized Cases summary. Conceptually, a SummarySectionDefinition provides a name and groups multiple paths linking to Attributes on the data layer. The simple user interface modeling allows declaring a grid layout to customize the user interface positioning to enable comprehensive summaries. Possible values for the position flag are: left, center, right, and stretched over the full width. Similarly, the TaskParamDefinition allows specifying a position flag which supports additional more complex values, such as left-center and center-right, that stretch over two columns. The purely meta-model-based approach binds each AttributeConstraint to exactly one representation on the user interface by default. To be compliant with the purely meta-model-based approach while allowing customizing where required, those representations might be overridden with CustomDataRepresentations. I.e., a string is interpreted as an SVG graphic or a JSON as a line diagram. Those CustomDataRepresentations can be extended according to domain-specific needs.

Most modeling elements declared on the case definition layer are instantiated by the case execution engine. The Case refers to the root Entity, the resolved owner and client path which each refer a User, the dedicated readers and writers that refer to Principals, the related Messages, SummarySections, and all Process elements. The current lifecycle is represented with a state property and the notes with a string property. Messages only exist on instance level and have properties, such as creation date, an author represented with a User, text content, and a seen by property listing all users that marked the message as seen.

Compared to a ProcessDefinition, multiple Processes are instantiated when a Process is repeated. The repetition order is expressed with the successor association. Each Process has a state property expressing the current state of the lifecycle process. The resolved CaseDefinition owner path refers to an Attribute and a flag indicating whether the linked owner is locally overridden on instance level.

During the case execution, Process specific technical Logs are created to enable debugging complex scenarios. A Log contains a creation date, a log level, a message, and an optional description property. Typically, the Logs are useful to debug the HttpHookDefinition execution representing the integration with third-party systems. The Alerts represent a slightly different concept for domain-specific issues which are relevant for the case-workers. According to the occurrence, three Alert types are distinguished. The case execution engine creates an error Alert when a hook execution fails and uses the custom declarable hook failure message if specified to provide end-user-friendly messages. For traceability reasons, a correction on a completed or terminated Task creates a correct Alert that illustrates the changes. Additionally, external custom Alerts can be created at any time by a third-party system to allow encapsulating complex domain-specific logic into external micro-services.

The Process inheritance structure on the case execution layer is analog to the case model definition layer. A TaskParam refers to the related TaskParamDefinition and the resolved path that links to an Attribute. The abstract TimedTask reference to an Attribute representing the resolved due date path and allows overriding the AttributeValue locally. The more complex lifecycle of a DualTask cannot be depicted with one state property. Therefore, two additional internal state properties are introduced to represent the human part and the automated part state lifecycle.

The meta-model contains several modeling elements with strongly coupled dependencies. Therefore, those composite associations are specified to support cascaded delete operations. I.e., a Case cannot exist without the associated CaseDefinition which contains substantial information that is needed for the Case execution. Therefore, when deleting a CaseDefinition, all associated Cases are deleted as well.

4.3. Execution Semantics

The case execution engine interprets the declared case-based process models. Therefore, this section highlights the crucial execution semantics defining the expected behavior of the case execution engine. Section 4.3.1 illustrates the underlying `Process` state machine, including possible state-dependent and state-independent transitions. Section 4.3.2 elaborates the instantiation of a declared `CaseDefinition`. Section 4.3.3 details the manual activation supported required for run-time planning. Section 4.3.4 illustrates the complete `Process` execution semantics and applies `SentryDefinition` evaluation. Similarly, Section 4.3.5 presents the `Process` termination. Section 4.3.6 describes the `SentryDefinition` evaluation in detail. Section 4.3.7 illustrates the `DualTask` internal state handling. Section 4.3.8 focuses on the `TaskParams`, which are neglected in the previous sections for reason of simplification.

4.3.1. Process State Lifecycle

The case execution engine is state-depend and therefore, the `Case` instance and all `Process` instances, i.e., `Stages`, `HumanTasks`, `DualTasks`, and `AutomatedTasks`, contain a state machine to indicate their current state. We used the CMMN 1.1. specification (Object Management Group, 2016, p. 113) as inspiration for our conceptual state and state transition design. Compared to the CMMN 1.1 specification, we reduced the number of states for reason of simplicity, whereas the expressiveness is mostly maintained. Respectively, the state transitions deviate. Figure 4.4 illustrates the possible process states, including their transitions. Our case execution engine supports the following `Process` states as briefly described:

- **Available** The `Process` is instantiated, but the precondition is not yet met for it to be enabled. I.e., the parent `Stage` is not active, or a required `SentryDefinition` is not yet satisfied, or a combination of both scenarios.
- **Enabled** The `Process` precondition is met, but the `Process` element needs a manual activation we use to support run-time planning. I.e., the parent `Stage` state is active and a possibly existing `SentryDefinition` is satisfied.
- **Active** The `Process` is currently performed, whereas the `Process` generated data is stored as a draft that is still modifiable. I.e., some `TaskParams` may be set, but the `Task` is not yet completed or terminated.
- **Terminated** The `Process` is unsuccessfully finalized. I.e., `Process` was maybe not suitable for this `Case` instance and the case-worker terminated that `Process`. Similarly, an external system may terminate an unsuitable `Process`.
- **Completed** The `Process` is successfully finalized. I.e., a case-worker performed all the required actions and completed the `Process`. Similarly, an external system typically completes `AutomatedTasks`.

Initially, every `Process` instance starts with the available state. If all conditions are met for the next state, the case execution engine automatically performs the state transition. The state description examples above assume that no intermediate automatic state transition follows. Compared to Processes, instantiated cases are automatically activated immediately after creation to allow working on the Case. The support of activation constraints for Cases is only applicable for nested Cases, which is currently not supported.

During a Task transition from enabled to active, the data binding is performed. Needed data structures are instantiated, which includes mostly an Entity with its Attributes. In the following, the TaskParams which link to the related Attributes are instantiated. The data binding is performed late to avoid an unnecessarily growing data structure with empty instances. Manual activation Tasks that may never be activated do not create any data object, similar to Tasks where not at least one SentryDefinition is satisfied.

During a Task transition from active to completed, the case execution engine validates all TaskParams. This includes the parameter type, multiplicity, and whether the parameter is mandatory. Multiple state transitions allow the direct termination of a Process from all states besides the completed state. Once a Process reached the final state completed or terminated, no state transitions are possible any longer.

The draft transition allows drafting TaskParams without affecting the Task state. This is especially usefully when a Task contains a TaskParam linking a DerivedAttribute to provide immediate feedback for the case-worker regarding the current DerivedAttribute value. It allows encapsulating the complex DerivedAttribute calculation within the case execution engine.

The correct transition allows correcting wrongly entered HumanTask or DualTask TaskParams representing a human input. During the correction, the TaskParams are validated to avoid violating data constraints. A successful correction does not lead to a reevaluation of Process constraints, such as a SentryDefinition. It is a design decision to prevent case-workers from creating unforeseen Process deadlocks. Linked data is updated, as usual, which includes the calculation of DerivedAttributes.

After a transition is performed, the declared HookDefinitions are executed, which enables process orchestration across system boundaries. A HookDefinition is specified to be executed post-transition while the previous Process or Case state might be synchronized with the previous transition. Additionally, the Case supports declaring a delete HookDefinition.

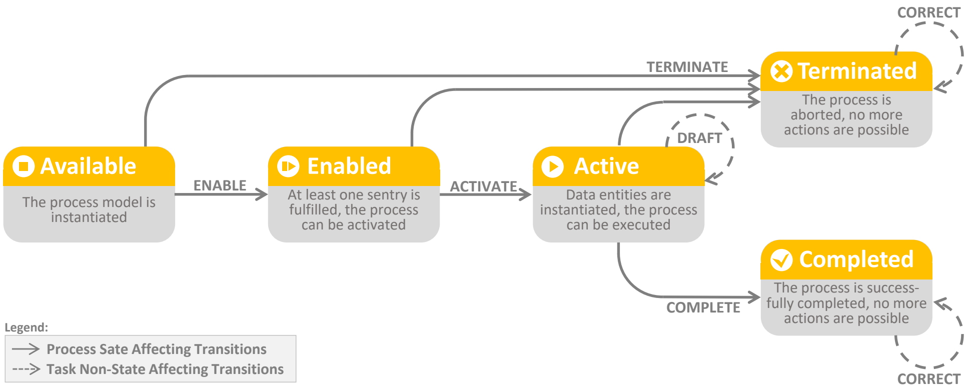

CORRECT
TERMINATE
Terminated
The process is aborted, no more actions are possible
DRAFT
Available
The process model is instantiated
ENABLE
Enabled
At least one sentry is fulfilled, the process can be activated
ACTIVATE
Active
Data entities are instantiated, the process can be executed
Completed
The process is successfully completed, no more actions are possible
COMPLETE
CORRECT
Legend:
Process Sate Affecting Transitions
Task Non-State Affecting Transitions

Figure 4.4.: Process state lifecycle adapted from Michel and Matthes (2018).

4.3.2. Instantiate CaseDefinition

A CaseDefinition is considered as a Case instance blueprint and has a method for instantiation, as presented in Listing 4.1. Outdated CaseDefinitions should not be instantiable. Therefore, a flag named isInstantiable indicates whether the CaseDefinition is instantiable or not. Additionally, the user who triggers the instantiation must be compliant with the constraints declared for the case owner. The case owner is declared with a linked Attribute that refers to the related AttributeDefinition that may have constraints on certain Groups. To prove that the owner constraint is fulfilled while the case data is not instantiated yet, AttributeDefinitions matching the linked owner path contained within the root EntityDefinition are checked. If the matched owner path is not found, the AttributeDefinitions contained within the optionally attached EntityDefinition after the root entity instantiation is checked similarly.

When the preconditions are fulfilled, the data layer is instantiated. First, the root EntityDefinition is instantiated as Case root Entity in the CaseDefinition workspace. In a second step, if an additional EntityDefinition is declared, this is instantiated similarly and attached to the root Entity according to the declared newEntityAttachPath variable.

After the data layer instantiation, the process layer instantiation follows. First, the Case is instantiated within the CaseDefinition workspace. The instantiated root Entity is bound to the Case. Additionally, the case meta-data linkage is performed as described in detail in Section 4.4.2. Therefore, the case ownerPath describing the path to the Attribute based on the Case root Entity is resolved and the owner is linked to the related Attribute. The same process is executed for the case client. Path resolving may lead to run-time exceptions when the path cannot be resolved based on the Case root Entity. Finally, the ProcessDefinitions are instantiated recursively, beginning with the direct children. During their instantiation, the Process state is automatically derived considering the parent Process, pending SentryDefinitions, and activation rules. A Task state change listener triggers instantiating and attaching the new EntityDefinition as optionally declared within the TaskDefinition when a Task becomes active.

```
CaseDefinition::public Case instantiate() throws EntityNotFoundException{

  if(!isInstantiableByCurrentUser())
    throw new CaseDefinitionIsNotInstantiableException();

  Entity rootEntity = new Entity(workspace, rootEntityDefinition);
  instantiateAndAttachNewEntityDefinition(rootEntity);

  Case case = new Case(workspace, this,  rootEntity, notesDefaultValue);
  case.initOwner(ownerPath);
  case.initClient(clientPath);

  for(ProcessDefinition pd : getDirectProcessDefinitions())
    pd.instantiate(case, null);

  return case;
}
```

Listing 4.1: CaseDefinition instantiation execution semantics.

4.3.3. Manually Activate Process

Certain ProcessDefinitions are modeled with a manual activation rule to enable run-time planning for knowledge workers. Those Processes only reach the state enabled and do not become automatically active when all sentries are satisfied. The knowledge worker must manually activate those processes before completing, which is described in the following. When manually activating a Process, the state is set to active, as illustrated in Listing 4.2. If the Process is an instance of a Stage, the sub Process might be enabled or activated as well. When the sub Process is depending on SentryDefinitions, at least one must be satisfied. SentryDefinitions are evaluated context-dependnt. Consequently, the Case and Stage are required for evaluation. The ProcessDefinition allows declaring a manual, an automatic and expression-based activation rule. Depending on the evaluation of the activation declaration, the Process state is set to enabled or directly to active. Considering that a sub Process might be an instance of a Stage, a recursive enabling or activation process is started. To enable or activate a nested Process, the parent Process must be active. Finally, when the process element is parallel repeatable, a new Process is instantiated according to the ProcessDefinition and enables knowledge workers to manually activate those Processes repeatedly. For serial repetitions, a new Process becomes enabled when the previous process is completed to avoid parallel instances.

```
Process ::public void manuallyActivate(){
  if(isEnabled()){
    state = State.ACTIVE;
    if(this instanceof Stage)
      enableOrActivateSubProcessesIfRequired((Stage)this);
    if(processDefinition.isParallelRepeatable())
      processDefinition.instantiate(case, parentStage);
  }else{
    throw new ProcessActivateException();
  }
}

Process ::private void enableOrActivateSubProcessesIfRequired(Stage stage){
  if(stage.isActive()){
    for(Process p : stage.subProcesses){
      if(p.processDefinition.isAtLeastOneSentryDefinitionSatisfied(p.case, stage)){
        if(p.processDefinition.isManualActivation())
          p.state = State.ENABLED;
        else
          p.state = State.ACTIVE;

        if(p instanceof Stage)
          p.enableOrActivateSubProcessesIfRequired((Stage)p);

        if(p.isActive() && p.processDefinition.isParallelRepeatable())
          p.processDefinition.instantiate(p.case, p.parentStage);
      }
    }
  }
}
```

Listing 4.2: Manual process activation execution semantics.

4.3.4. Complete Process

When completing a Process, multiple dimensions must be considered, such as the Process element to be completed, all sub Processes, the parent Stage if applicable, and SentryDefinitions if applicable. Listing 4.3 presents the complete entry point. As a precondition, the current Process state active must be ensured. Otherwise, the Process completion is aborted with an exception. Next, all mandatory sub Processes must either be completed or terminated to proceed. All not yet completed or terminated non-mandatory sub Process are automatically recursively terminated. Distinguishing mandatory and not mandatory Processes combines explicit decisions from knowledge workers for mandatory Processes and allows simplified implicit decisions for non-mandatory Process. After all preconditions are met, the Process state is changed to complete. It must be determined if a new Process may be instantiated or an existing Process may be updated. Finally, if applicable, the parent Stage is completed. For reason of simplification, the TaskParam validations applied before each Task completion is neglected, which is conceptually described in Section 4.3.8.

```
Process ::public void complete(){
  if(!state.isActive())
    throw new ProcessCompleteException();
  if(this instanceof Stage) {
    for(Process sp: ((Stage)this).subProcesses)
      if(sp.processDefinition.isMandatory&&(sp.isAvailable()||sp.isEnabled()||sp.isActive()))
        throw new SubProcessCompleteException();
    for(Process sp : ((Stage)this).subProcesses)
      if(!sp.processDefinition.isMandatory&&(sp.isAvailable()||sp.isEnabled()||sp.isActive()))
        sp.terminate();
  }
  state = State.COMPLETED;
  mayInstantiateOrUpdate();
  mayAutocompleteParentStage();
}
```

Listing 4.3: Process completion execution semantics.

Listing 4.4 determines whether an instantiation or an update is required, or no change is applicable. The completion of a Process may partly satisfy a SentryDefinition depending on multiple Processes or fully satisfy a SentryDefinition depending only on this Process. A fully satisfied SentryDefinition enables an additional Process. Therefore, an iteration over all potentially satisfied SentryDefinitions evaluates whether one or more SentryDefinitions are satisfied. It is important to note that SentryDefinitions containing recursions must be checked last. To evaluate whether a SentryDefinition is satisfied, the case and parent Stage are passed as context, whereas the parent Stage is null if not applicable. Section 4.3.6 describes the SentryDefinition evaluation in detail. When a SentryDefinition is evaluated as satisfied, the enabled Process is updated or, if needed, a new Process is instantiated. When a Process does not depend on a SentryDefinition, the completion may lead to an additional Process instantiation when repeatable.

```
Process::protected void mayInstantiateOrUpdate(){
  for(SentryDefinition sentryDefinition : processDefinition.
    getSatisfyingSentryDefinitionsOrdered())
    if(sentryDefinition.isSatisfied(case, parentStage)){
      ProcessDefinition enablesPd = sentryDefinition.enablesProcessDefinition;
      Process enablesP = sentryDefinition.getEnabledProcessIfInstantiated(case, parentStage);
      instantiateOrUpdate(enablesP, enablesPd);
    }
  if(!processDefinition.hasEnablingSentryDefinitions())
    instantiateOrUpdate(this, processDefinition);
}
```

Listing 4.4: Process checking instantiation or update execution semantics.

Listing 4.5 illustrates the instantiation or update of a `Process`. The first condition checks if a new `Process` instantiation is required. Either there is no `Process` instance, or the current instance is completed or terminated while the `ProcessDefinition` is repeatable. For the `Process` instantiation, the parent `Stage` is required as contextual information. Considering that the variable p may be null, the parent `Stage` must be resolved with the `Case` and `StageDefinition` context if applicable. Subsequently, the new `Process` is instantiated based on the `ProcessDefinition` and the `Case` and parent `Stage` context. The second condition enables an existing `Process` if the current state is available. Considering that the `Process` represents a `Stage`, sub `Processes` are checked for state updates and new `Process` instantiations.

```
Process::private void instantiateOrUpdate(Process p, ProcessDefinition pd){
  if(p==null||(p.isCompleted()||p.isTerminated())&&(pd.isSerialRepeatable()||pd.
    isParallelRepeatable())){
    Stage parentStage = null;
    if(pd.hasParentStageDefinition())
      parentStage = (Stage) pd.parentStageDefinition.getProcess(case, this.parentStage);
    p = pd.instantiate(case, parentStage);
  }else if(p.isAvailable())
    p.initState();
  if(p instanceof Stage)
    for(Process subProcess : ((Stage)p).subProcesses)
      subProcess.mayInstantiateOrUpdate();
}
```

Listing 4.5: Process instantiation or update execution semantics.

Listing 4.6 determines the possible automatic completion of parent `Stages`. To be applicable, the `Process` must have a parent `Stage`. All sub `Processes` of the parent `Stage` must be completed or terminated. The state is changed to completed and it must be check if a process instantiation or update may be required. Finally, the parent `Stage` completion is applied recursively until the root `Stage` is reached.

```
Process::protected void mayAutocompleteParentStage() {
  if(hasParentStage()) {
    for(Process process: parentStage.subProcesses)
      if(!(process.isCompleted() || process.isTerminated()))
        return;
    parentStage.state = State.COMPLETED;
    parentStage.mayInstantiateOrUpdate();
    parentStage.mayAutocompleteParentStage();
  }
}
```

Listing 4.6: Process that can autocomplete parent stage execution semantics.

4.3.5. Terminate Process

Adaptive case management is a knowledge-intensive endeavor. Hence it is not possible to successfully complete all `Processes` as planned. Therefore, the possibility to terminate a `Process` is provided, as illustrated in Listing 4.7. Technically, all `Processes` which are not yet in the state completed or terminated can perform the termination operation. When the `Process` to be terminated is an instance of a `Stage`, all sub `Processes` are terminated recursively first, and then the state is set to terminated.

```
Process::public void terminate(){
  if(!isCompleted() || !isTerminated()){
    if(this instanceof Stage)
      for(Process sp : ((Stage)this).subProcesses)
        sp.terminate();
    state = State.TERMINATED;
  }else{
    throw new ProcessTerminateException();
  }
}
```

Listing 4.7: Process termination execution semantics.

4.3.6. Satisfy SentryDefinition

Certain `Processes` depend on results from other `Processes`. Therefore, `SentryDefinitions` allow expressing preconditions on other `Process` elements or expressions on the resulting data structure, as illustrated in Listing 4.8. Each `SentryDefinition` refers to at least one or more `Processes` that must be completed to satisfy the `SentryDefinition` and enable the related `Processes`. One `ProcessDefinition` might have multiple `SentryDefinitions` assigned, whereas only one of those must be satisfied to enable the related `Process`.

Compared to most other meta-model elements, `SentryDefinitions` are not instantiated. To evaluate whether a `SentryDefinition` is satisfied, a mapping between the process and the process-definition layer is required. Depending on the declared meta-model, an evaluation might be expensive. Evaluations are only triggered when a referenced process of a `SentryDefinition` is completed, which could satisfy those. The method that determines whether a `SentryDefinition` is satisfied needs as contextual information the `Case` instance and `Stage` if applicable. The `Stage` is null when the `ProcessDefinition` to which the `SentryDefinition` is assigned, is on the root level. First, a map between `ProcessDefinitions` and the last instantiated `Process` elements within the `Case` context and the `Stage` context is calculated. To prove whether the `SentryDefinition` is satisfied, an iteration over all `ProcessDefinitions` preconditions that must be completed is started. A map lookup returns the related `Process` if instantiated and checks whether the state is completed. The evaluation ends with the first occurrence of a not found or not yet completed `Process`. If all pending `Processes` are completed and there no additional expression is declared, the `SentryDefinition` is satisfied. When an expression is declared, this is evaluated based on the case root entity and decides whether the `SentryDefinition` is satisfied or not.

Introducing parallel repetitions is accompanied by the limitation that SentryDefinitions are only declarable within the same ProcessDefinition nesting level. Otherwise, a StageDefinition that is parallel repeatable and contains two TaskDefinitions having a dependency declared with a SentryDefinition could not be evaluated as expected. Considering that both repetitions are currently in the state where no Task is completed and now one Task gets completed, the mapping could randomly lead to the activation of the Task within the other Stage, because both are referencing the same TaskDefinition.

```
SentryDefinition::public boolean isSatisfied(Case case, Stage stage){
  Map<ProcessDefinition, Process> map = case.calcProcessDefinitionToProcessMap(stage);
  for(ProcessDefinition processDefinition : satisfyingProcessDefinitions)
    if(!map.containsKey(processDefinition) || !map.get(processDefinition).isCompleted())
      return false;
  if(hasExpression())
    return case.evaluateExpression(expression);
  return true;
}
```

Listing 4.8: SentryDefinition satisfaction execution semantics.

4.3.7. DualTask Internal State Handling

Conceptually, a DualTask combines a HumanTask followed by an AutomatedTask and therefore, two states are internally maintained. Aggregated, those states represented the global DualTask state similar to all other Process elements. Table 4.1 presents the global state and the related human part and automated part states. Enabling a DualTask enables the human part as well, while the automated part is still available because the human part is not yet completed. Activating a DualTask affects similarly only the human part state. Completing the human part activates the pending automated part. A manual activation of the automated part is conceptually meaningless, and therefore, the enabled state is skipped. Completing the automated part completes the overall DualTask, similar to the termination. Conceptually, the DualTask internally uses the generic Process execution logic and extends it to manage the two internal states. The Process state change interfaces for manual activation and termination are compatible with the generic interface, while the complete interface needs additional information on which part to be completed.

	DualTask state	
Global	**HumanPart**	**AutomatedPart**
Available	Available	Available
Enabled	Enabled	Available
Active	Active	Available
	Completed	Active
Completed	Completed	Completed
Terminated	Completed	Terminated
	Terminated	Terminated

Table 4.1.: DualTask global state and internal states.

4.3.8. Modifying Task Parameters

Unlike the execution semantics described in Sections 4.3.2 - 4.3.7, the execution semantics modifying `TaskParams` does not necessarily lead to a `Process` state change. Depending on the `TaskDefinition`, different operations are applicable that modify the `TaskParams` and the resulting `Case` data respectively, as illustrated in Table 4.2. Draft operations are applicable for all `Task` types, considering the precondition that the `Task` state is active. Accomplishing a draft operation does not affect the state. Typically, draft operations are used to update writable `TaskParams` to receive updated read-only `TaskParams` pending on `DerivedAttributes` which allows providing user feedback instantly. The meta-model allows declaring `TaskParamDefinitions` as mandatory and linked `AttributeDefinition` allows declaring a multiplicity. When a `Task` draft is performed, not all mandatory `TaskParams` may be declared and linked attributes multiplicity may be violated. Besides this meta-model consistency violation, all other constraints are fulfilled, which leads to a mostly consistent meta-model. Complete operations are applicable for all `Task` types, whereas the `Task` state is changed from active to completed. Correct operations meant to correct human input errors and therefore only applicable for `HumanTasks` or the human part of `DualTasks`. All operations support human-readable error messages for parsing errors or meta-model validation errors. Figure 4.4 indicates possible operations depending on the `Process` state context. The `TaskParam` modification implicitly updates the linked `Attribute`. Therefore, other meta-data fields linking this attribute are implicitly updated as well if not explicitly overridden. I.e., an `Attribute` representing a `Case` role is updated, which implies that all `Tasks` referencing those `Attributes` as an owner are implicitly updated as well, unless the owner is manually overridden for a task.

		Draft	**Complete**	**Correct**
Applicable	HumanTask	✔	✔	✔
	AutomatedTask	✔	✔	✘
	DualTask (HumanPart)	✔	✔	✔
	DualTask (AutomatedPart)	✔	✔	✘
Supports	Idempotence (no Process state change)	✔	✘	✔
	Parsing error messages	✔	✔	✔
	Meta-model validation errors messages	✔	✔	✔
	Mostly consistent with meta-model	✔	✔	✔
	Fully consistent with meta-model	✘	✔	✔
	Dynamic on-the-fly updates	✔	✘	✘

Table 4.2.: Task operations that modify task parameters and resulting case data.

4.4. Conceptual Design Challenges

In this section, we highlight conceptual relevant design challenges and present the related design decisions. Case management is composed of data that represents the artifacts and processes that describe possible activities depending on the current artifacts. Therefore, Section 4.4.1 illustrates the crucial linkage between process and data layer. Processes elements describe activities with several linked meta-data fields. Section 4.4.2 illustrates the linkage between the meta-data-fields and the data layer. The relation between meta-data fields, roles, and dedicated case access rights are presented in Section 4.4.3. Occurring attribute multiplicity challenges are illustrated in Section 4.4.4. The complexity of interoperability models is shown in Section 4.4.5, and the challenges regarding the interoperability of non-model-based systems are discussed in Section 4.4.6. Dealing with human input errors in the context of adaptive case management is illustrated in Section 4.4.7. Finally, the generic reusable representation and the needed customization is discussed.

4.4.1. Linkage between Process and Data Layer

A case management system needs a conceptual process layer that orchestrates the data generation and a conceptual data layer that stores the resulting data. Therefore, the ensemble acting between process and data layer is crucial. The CMMN specification (Object Management Group, 2016) describes the `CaseFile` as information context where a `Task`'s in- and outputs can be referenced. This specification facilitates reusing a `Task`'s output as input for a following `Task`. Embedding the resulting data into `Tasks` prevents the following `Tasks` referencing the previously generated data and cannot be considered as an expressive solution.

Therefore, a separated process and data layer are needed where a task can reference previously generated case data and enriches case data with new input. Accompanied by the increasing expressiveness, the complexity of such an approach is rising significantly. In Section 4.2, we presented our conceptual meta-model and in the following, we will illustrate occurring challenges based on a sample with an object diagram. The simple sample contains a repeatable `Stage` that contains a mandatory `HumanTask` as illustrated in Figure 4.5, and the related conceptual object diagram is illustrated in Figure 4.6. The object diagram merely shows objects and associations relevant for the data linkage and applies the coloring schema similar to the meta-model.

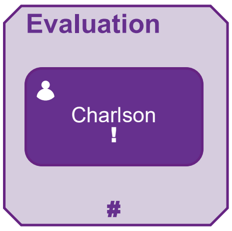
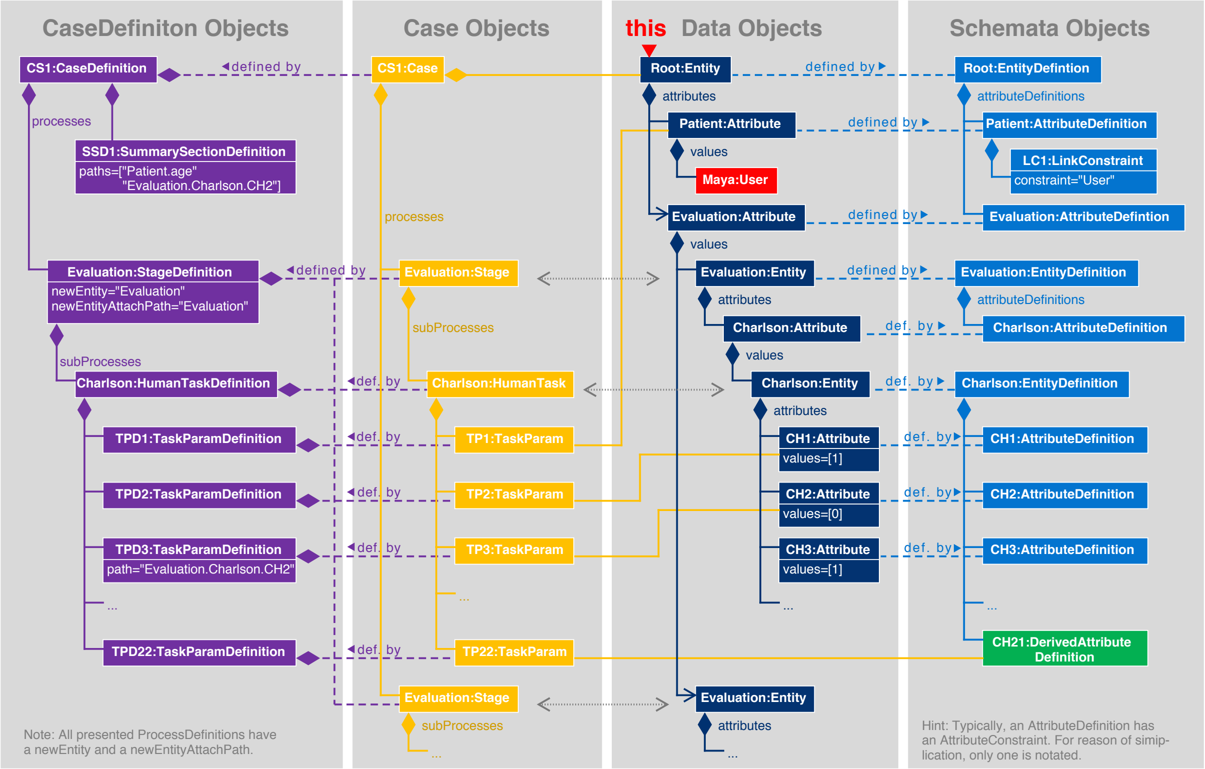


Figure 4.5.: Sample notated in adapted CMMN.



Note: All presented ProcessDefinitions have a newEntity and a newEntityAttachPath.

Hint: Typically, an AttributeDefinition has an AttributeConstraint. For reason of simiplication, only one is notated.

Figure 4.6.: Instantiated meta-model representing a `Stage` containing a `HumanTask` that illustrates the linkage between processes and data structures. For reason of simplification, only important objects and associations are visible.

Each `Case` has exactly one root `Entity` to store all generated related data. The `CaseDefinition` specifies the `EntityDefinition` which is used to instantiate the root `Entity`. The root `Entity` uses `Attributes` to either store simple values or to reference to `Entities` representing a complex value. The referencing approach leads to a hierarchical tree-based data structure for each `Case`. To allow data re-usage, data references are declared with paths starting on the `Case` root `Entity`. In the object diagram, a red `this` highlights the `Case` paths base reference.

`Processes` do not need any data access until they reach the state active. Late data binding is applied to keep the resulting data structure lean. Upon `Process` activation, data binding is performed. First, the `Case` data structure is extended if declared in the `ProcessDefinition` and second, if the `Process` is an instance of a `Task`, the `TaskParam` binding is performed. The `newEntity` `ProcessDefinition` attribute references an `EntityDefinition` that should be attached and a `newEntityAttachPath` specifies where the instantiated `EntityDefinition` should be attached as `Entity`.

I.e., activating the `HumanTask` Charlson leads to a Charlson `EntityDefinition` instantiation that is attached as `Entity` on the path `"Evaluation.Charlson"`. Path resolving uses the `Attribute` names and considers the last `AttributeValue` as reference to support repetitions. Nested paths are separated with dots and the resolving is applied for each level until the path is completely resolved. With gray arrows, the object diagram indicates which process activation triggers attaching entities. After the case data extension, the `TaskParam` binding follows. Typically, most `TaskParams` reference `Attributes` of the newly attached `Entity` and partially, `TaskParams` reference to existing `Attributes` to reuse data. The paths for the `HumanTask` Charlson `TaskParams` are declared as following:

```
"Patient.age"
"Evaluation.Charlson.CH1"
"Evaluation.Charlson.CH2"
"..."
"Evaluation.Charlson.CH21"
```

The first path references an existing attribute, the following two reference to the newly created `Entity` `Attributes` and the last path references to a `DerivedAttributeDefinition` belonging to the `EntityDefinition` of the newly created `Entity`. `DerivedAttributeDefinitions` are not referable on instance level due to the dynamic computation. Therefore, the `DerivedAttributeDefinition` is referenced, which can compute the actual value with an additional reference to the `Entity`. All resolved `TaskParamDefinition` path references are materialized on the `TaskParam` to ensure a linear access time and to prevent increasing complexity due to `Task` repetitions.

The referencing approach allows creating data structures that deviate from the process structure. However, when `Process` elements are repeated, typically a new `Entity` is attached and used as `TaskParam` reference to encapsulate the repetition, with the exception of `TaskParams` that only reference existing `Attributes`, which is useful to show read-only contextual information or to update existing `Attributes`. The referencing of the patient's age is a sample for contextual read-only information and updating a role when a certain stage is activated would be a sample for an updatable `Attribute`.

A `SummarySectionDefinition` declares paths similar to `TaskParamDefinition` with the difference that those paths are dynamically resolved on each request. Considering a repeatable `Process`, the resolved `SummarySectionDefinition` paths automatically point to the last `Process` iteration which is desired. The access time depends on the path depth, however, `SummarySections` provide a read-only view which is only accessible for the end-user, which if needed could be computed technically in parallel.

The instantiated `CaseDefinition` objects contain the repeatable declaration and the `Case` objects are instantiated for each repetition. I.e., the repeatable Evaluation `Stage` is instantiated multiple times, as indicated at the bottom of the object diagram. A similar pattern is applied for the schemata objects and the data objects. The object diagram illustrates one instantiated case, but all `CaseDefinition` objects and `Schemata` objects are used for multiple `Case` instances.

Resolving paths during run-time may lead to run-time errors. Model-based consistency checks can reduce the probability but cannot prevent all occurring exceptional cases. I.e., wrongly modeled `Attribute` multiplicities can prevent attaching a new `Entity` and prevent a planned repetition. To summarize, the separated data and process structures in combination with the repetitions and the run-time path resolving is expressive but also challenging.

4.4.2. Meta-Data Linkage

Our process layer supports different meta-data fields that are typically static, such as the CaseDefinition version and modifiable meta-data files, such as the Case state or owner. Typically, meta-data files are modifiable on the containing element itself. Considering that a Task owner would only be modifiable on their Task, much effort is needed to keep the owner meta-data field of all Tasks up to date. Having up-to-date meta-data information helps clarify needed contributions and enhance the collaboration. I.e., a Task owner can be notified about overdue Tasks, provided the latest meta information is available.

Therefore, our approach links certain meta-data fields to Attributes on the data layer which enables using process-generated data as meta-data. Multiple modeling elements can link to one Attribute allowing to batch update all related meta-data fields. Modeling instance elements supporting linked meta-data fields are:

Case	
↳ owner	Responsible to manage the case.
↳ client	In the medical context, typically the patient.
Process	Applicable for all inherited modeling elements as well.
↳ owner	User responsible to complete the process (supports overriding).
↳ dynamicDescription	Dynamically extends the task title with an AttributeValue.
TimedTask	Applicable for a HumanTask or a DualTask.
↳ dueDate	Defines a due date (supports overriding).

The meta-data field binding uses path declarations that are resolved during run-time as described in Section 4.4.1. All declared paths are resolved directly after instantiation and conceptually linked with a change listener concept. If the referenced Attribute changes, the meta-data field is generally updated as well. For traceability reason, the Process owner and TimedTask due date field is only updated until the Process is completed or terminated, further changes are not propagated into the meta-data fields. Typically, several Tasks linking to an identical Attribute as owner or due date. Practical, those Attributes represent a role. Predefined Process owners are helpful to enable collaboration but may need to be adapted independently from other Processes linking the identical Attribute. Therefore, the due Process owner and TimedTask due date field support to locally overriding the linked AttributeValue. Besides the Case owner and client meta-data field, the meta-data linkage is desired but optionally. Without linking the local overriding mechanism is still applicable.

4.4.3. Dynamic Roles and Dedicated Access Rights

In the context of the medical domain, dedicated case access rights are essential to ensure data privacy. However, the best conceptual design principles are worthless if they are not applicable in practice. Therefore, the ACM4IC approach combines inherited case access rights from workspaces, automatically grants case write access for users assigned to case roles, and enables explicit granting and removing case access rights.

All Attributes referenced as owner meta-data fields from at least one Process are implicitly interpreted as case roles. A case role is implicitly declared with the Process owner path. The modeler ensures that the linking AttributeConstraint only approves Users who are applicable for that role. Within a LinkConstraint, multiple Groups might be declared where the set of all members represents valid values. When manually overriding the Process owner, the LinkConstraint must still be fulfilled.

A Process owner needs case write access to perform expected contributions. Therefore, when a process owner is changed, write access rights are automatically granted if needed. This applies for changes triggered from a linked attribute or manual overriding. Conceptually, granting access rights and changing the role could be separated into two steps. However, the end-user would need to know the User roles when granting access rights to decide whom to grant write access. Considering that a case has multiple roles and Users might be members of multiple Groups, this could be confusing. To ensure a maximum degree of flexibility, all users with WRITE access regardless of the owner meta-data field can perform Task actions. All performed Task actions are logged and shown for reason of traceability. This design principle allows to quickly take over tasks without any overhead, e.g., from ill colleagues.

Additionally, role-independent access rights may be granted or revoked at any time. Access levels such as READ and WRITE may be granted to dedicated Users. The access level CASEOWNER may be granted to exactly one User who is responsible for managing the case. The Case owner meta-data field and the CASEOWNER access rights are synchronized. A Case owner has dedicated access rights to complete, terminate, or delete a Case. In addition to the changeable access rights, the concept of inherited access rights allows granting access to all Cases of a Workspace on the READ or WRITE access level. I.e., inherited access could be used for administrative purposes.

4.4.4. Consistent Attribute Multiplicity

An AttributeDefinition allows to declare the expected multiplicity and uses as the default value the multiplicity any. Typically, TaskParamDefinitions link to AttributeDefinitions modeled with the multiplicity exactlyOne. Before completing a task, all task parameters are validated, which includes the validation of the multiplicity constraint. Only successfully validated tasks are completed. TaskParamDefinition may link to a DerivedAttributeDefinition that automatically computes values based on an expression. The expression is declared in the model and typically aggregates information from other AttributeDefinitions belonging to the identical EntityDefinition. The expressive power allows to declare very complex expressions which must be evaluated in the backend.

Usability considerations desire immediate feedback for user input which means that on an Attribute change, the related DerivedAttributes must be updated on the fly. Drafting task values allow providing immediate feedback. However, a task that contains multiple Attributes and a DerivedAttribute can typically not fulfill the exactlyOne multiplicity constraint after the first change event. The described issue does not occur if default AttributeValues are specified. Therefore, the task draft validation slightly allows violating the multiplicity constraint. If no AttributeValue is set, the multiplicity constraint violation is ignored; if an AttributeValue is set, the multiplicity constraint is regularly validated to prevent having inconsistent data. The illustrated approach is a practical trade-off between usability and consistency considerations.

4.4.5. Complex Interoperability Models

In the context of integrated care, multidisciplinary professional teams work collaboratively to provide patient-centered care. The seamless incorporation of patient self-management functionalities, such as monitoring is crucial success factors. Today, a typical heartbeat monitoring is performed during a patient visit in the hospital. Integrated care aims to incorporate such features in the patient's case to enable long-term monitoring before or after hospitalization.

The conceptual flow is executed as follows: First the clinical professional completes a `HumanTask` to specify the monitoring type, the daytime when to perform the measurement, a start date, and an end date. Second, a pending `AutomatedTask` uses the `HumanTask` selected parameters to orchestrate the patient's monitoring actions with a third-party system. The `AutomatedTask` collects the time-series data until the monitoring prescription ends. Monitoring prescriptions are created based on the needs of the individual patient and therefore, they are manually activated during the case execution. Figure 4.7 illustrates the described scenario on the left side.

Considering our aim to provide a holistic model-based system which includes the frontend and backend implementation, the provided sample would lead to crucial usability issues in the frontend. A generic representation of the notated sample leads to three elements on the user interface, where from the clinical point of view, all this information is strongly related and should be accessible on one screen. Therefore, we introduced the generic `DualTask` concept that combines a `HumanTask` followed by an `AutomatedTask` as illustrated on the right side in Figure 4.7. This simplification enables representing all `TaskParams` on one screen, to correct the `HumanTask` `TaskParams` if needed in the context of the measurements, and it additionally simplifies the technical integration.

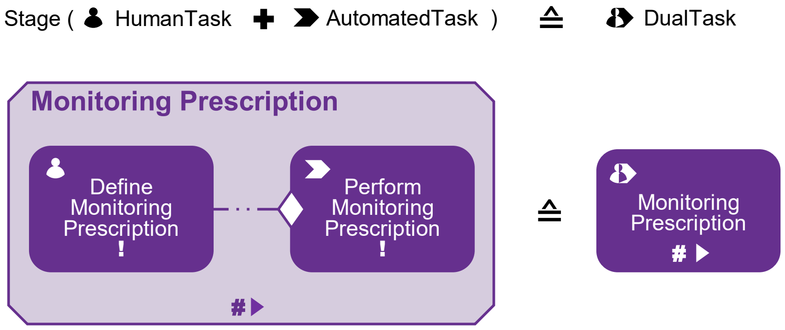

Stage (HumanTask + AutomatedTask) ≙ DualTask
Monitoring Prescription
Define Monitoring Prescription
!
Perform Monitoring Prescription
!
# ▶
≙
Monitoring Prescription
# ▶

Figure 4.7.: A monitoring prescription expressed with a `Stage` containing a `HumanTask` followed by an `AutomatedTask` or expressed with a single `DualTask`.

4.4.6. Interoperability with Non-Model-Based Systems

Notably, in the integrated care context, process orchestration across system boundaries and organizational boundaries is crucial. A combination of different model-based approaches enables such a seamless integration with typically non-model-based third-party systems.

The CMMN specification (Object Management Group, 2016) defines `EventListeners`, which are listening to `Stage` and `Task` intending events, that are either time-triggered or user-triggered. In combination with `Sentries`, these `EventListeners` allow to define external dependencies. However, CMMN does not specify notification mechanisms that allow orchestrating external systems. Therefore, we introduced the `HttpHookDefinition` concept, which allows to declare notifications on state change events for a `Stage` or for any `Task` type. The declaration contains a URL, the HTTP method, the state transition event, a failure message, and a flag indicating whether the `Process` element should be attached as a serialized payload. If the execution fails, an error `Alert` is automatically generated and attached to the `Process` element. We notated an `HttpHookDefinition` as an additional decorator for any `Process` element illustrated in Figure 4.8. The `HttpHookDefinition` concept allows sending information to external systems and the default API endpoints allow receiving information from external systems which enables a full process orchestration.

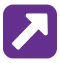


Figure 4.8.: `HttpHookDefinition` decorator notation.

The interfaces, which represent the model-based to non-model-based transition, face challenges to deal with the fracture of design principles. Interfaces of non-model-based systems are more flexible regarding adaptations during design-phase, but less flexible during the production phase. Interfaces of a model-based system provide much flexibility within their meta-model capabilities, but adaptations beyond are challenging. We Assume that non-model-based monitoring prescription services should be integrated into a case management system that is used in different organizations with slightly different needs. A possible solution is to customize the integration to every organization individually or to provide a generic solution that might be reused which is preferable to keep the integration effort comparably low. Each organization is highly likely to require unique `CaseDefinition` models representing customized workflows with `ProcessDefinitions`. To allow specifying custom models while using the similar service interfaces for integration, our approach supports declaring an `externalId` which allows the external unified service to map all customized models to an underlying more abstract model. Typically, an `externalId` is declared for a `ProcessDefinition`, and their related `AttributeDefinition` and for each attribute `EnumerationConstraint` if needed. Additionally, an `externalId` can also be assigned to the related instantiated objects to support dynamic mapping.

4.4.7. Dealing with Human Input Errors

In the medical context within a hospital where professional case-workers perform knowledge-intensive tasks, such as completing a medical questionnaire in cooperation with a patient, errors may occur. Rasmussen and Vicente (1989, p. 517) state "that reliable human-system interaction will be achieved by designing interfaces which tend to minimize the potential for control interference and support recovery from errors. In other words, the focus should be on control of the effects of errors rather than on the elimination of errors per se."

Medical questionnaires are typically modeled as tasks in CMMN. According to the CMMN specification (Object Management Group, 2016, p. 113), completed or terminated tasks are immutable. Technically, the specified behavior is desired to prevent undesired side effects. However, from a medical perspective, human input errors must be correctable, because it is unlikely to prevent errors completely. CMMN supports the declaration of tasks as repeatable, which allows creating a second task instance with the corrected input under certain circumstances, such as when the parent stage is still not completed or terminated. The declaration of repetitions to correct human input errors leads to a more complex model which has a conceptually different meaning and most likely confuses the case-worker.

Assuming that most errors consist of a single wrongly set `TaskParam`, our approach supports a more lightweight and intuitive correct functionality. Completed or terminated tasks allow correcting any `TaskParams` with their underlying `Attributes`, may be affected `DerivedAttributes` are dynamically updated. For reasons of traceability, a correction creates an `Alert` for the related `Task` indicating the manual adjustments. An `Alert` indicates when the correction was performed by who and lists the affected `TaskParams` with their old and new values. Correcting a `HumanTask` does not change any task's state and does not trigger any sentry evaluation.

Figure 4.9 illustrates a pre- and post-correction example illustrating a hypothetical border case. The pre-correct state shows a `HumanTask` that represents a Charlson Comorbidity Index (Charlson et al., 1987) questionnaire that results with the final score two. A `Sentry` satisfied with a Charlson score below five activated `Task A` that was completed afterwards. Now we assume that `Task A` satisfies another sentry, thus enabling additional `Processes`. Now, the Charlson `HumanTask` is corrected so that the underlying `DerivedAttribute` dynamically changes the score to seven. Theoretically, the sentry for `Task B` would be satisfied and `Task A` should be not enabled. However, the sentry evaluation is not triggered after a correction, because unforeseen deadlocks could occur, and performed actions cannot be undone with all pending `Processes`.

Our solution presents an additional opportunity to correct simple human input errors intuitively. The correct functionality is applicable for most of the relevant practical samples we noticed. However, there might be cases where a re-execution is necessary, which still can be modeled and decided by the case-worker.

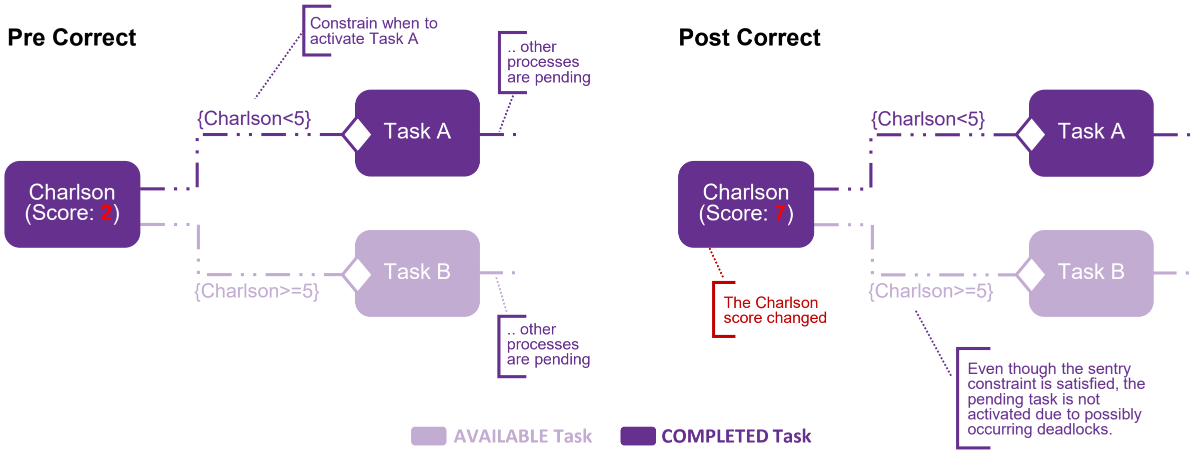

Pre Correct
Constrain when to activate Task A
.. other processes are pending
{Charlson<5}
Task A
Charlson (Score: 2)
Task B
{Charlson>=5}
.. other processes are pending
Post Correct
{Charlson<5}
Task A
Charlson (Score: 7)
Task B
{Charlson>=5}
The Charlson score changed
Even though the sentry constraint is satisfied, the pending task is not activated due to possibly occurring deadlocks.
AVAILABLE Task
COMPLETED Task

Figure 4.9.: Conceptual correct example notated in adapted CMMN.

4.4.8. Generic Reusable Representation vs. Customizability

Entirely meta-model-based approaches focusing on data and process modeling are typically using generic data and process representations. Each data field type is mapped to precisely one representation, including a depiction to modify the field and to show a read-only view. This approach requires only a minimum set of representations and enables a maximum re-usability.

However, as Mayhew (1999, p. 1) states: "The user interface to an interactive product such as software can be defined as languages through which the user and the product communicate with each other. [...] As far as users are concerned, the user interface is the product." Therefore, tailoring user interfaces is a critical success factor to increase end-users acceptance.

Typical solutions use a process engine for process management and individually develop a Single Page Applications (SPA) to provide customized use case-specific representations. This approach is only sufficient for large scale process with many instances. Our approach combines generic reusable and customization aspects into a purely meta-model-based concept.

Every `AttributeConstraint` is bound to exactly one generic readable and editable representation by default. If needed, the default representation is overridden with a `CustomDataRepresentation` that allows interpreting the data differently. I.e., a dynamic calculated string might be interpreted as a SVG graphic or JSON data might be interpreted as a diagram. The combination of both concepts allows covering most use cases with the generic reusable representation, but allows providing custom tailored solutions when actually needed. The detailed meta-model (cf. Figure A.1) lists all possible `CustomDataRepresentations` and related samples are presented in the prototypical implementation (cf. Section 5.1.9).

4.5. Supported CMMN Elements

In Section 2.1.2, we presented the CMMN 1.1 specification, which we used as a reference for our conceptual design. In this section, we present the mapping between our conceptual modeling elements and the CMMN specification. Our conceptual design uses an existing data-centric meta-model as a base which was extended to support adaptive case management. To provide a consistent terminology within the meta-model, we slightly adapted the naming used in the CMMN specification.

Supported elements, including their decorator applicability, are illustrated in table 4.3. All model elements used for declaration contain a name postfix `Definition`, whereas the related instances do not have a postfix. I.e., a `CaseDefinition` or colloquial named case template can be instantiated and the corresponding instantiated element is named `Case`. The `EntityDefinition` is comparable with the CMMN `CaseFile` schemata to define the case data structure.

Our aim is to holistically support adaptive case management scenarios for integrated care applications. Therefore, we carefully analyzed the needed CMMN elements. Rolling run-time

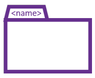
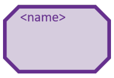
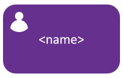
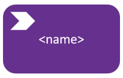
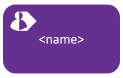
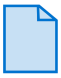
Decorator Applicability	Entry Criteria ◇	AutoComplete ■	Manual Activation ▶	Required !	Repetition #	Hook* ↗
`CaseDefinition` 	✘	✔	✘	✘	✘	✔
`StageDefinition` 	✔	✔	✔	✔	✔	✔
`HumanTaskDefinition` 	✔	✔	✔	✔	✔	✔
`AutomatedTaskDefinition` 	✔	✔	✔	✔	✔	✔
`DualTaskDefinition`* 	✔	✔	✔	✔	✔	✔
`EntityDefinition` 	✘	✘	✘	✘	✘	✘

Table 4.3.: Supported modeling elements including their decorator applicability, where the CMMN notation is slightly adapted to be compliant with our conceptual layer coloring schema. Modeling elements that are extending the CMMN 1.1 specification are indicated with a star.

planning is implicitly supported with manual activation `Tasks`. A case-worker can add an enabled manual activation `Task` at any time. Compared with a CMMN planning table, the primary limitation is that multiple tasks must be added sequentially instead within one step. Considering that the clinical run-time planning is mostly performed ad-hoc depending on the current context, this should be manageable. The autocomplete decorator is applied implicitly on all `Stages` but cannot be modeled explicitly. As presented with the conceptual design challenges in Section 4.4, we introduced additional modeling elements such as a `DualTask` and a `Hook` decorator that extend the CMMN notation.

4.6. Summary of Conceptually Supported Requirements

In Chapter 3, the requirements are derived from the literature, while this chapter details the conceptual implementation of the requirements. Table 4.4 lists conceptual layers representing corresponding meta-model elements as rows and lists the requirements as columns. The resulting matrix illustrates which conceptual layer supports a requirement. A requirement is considered as partly supported if the requirement is not fully satisfied by a conceptual layer or the corresponding meta-model elements. The three high-level requirements are considered as fully supported if all subordinate requirements are supported. The last row indicates the aggregated conceptual approach fully supporting the requirements.

		R1	R1.1	R1.2	R1.3	R1.4	R2	R2.1	R2.2	R2.3	R3	R3.1	R3.2	R3.3	R3.4	R3.5
⬛	Simple UI Models	◐	○	○	○	●	○	○	○	○	○	○	○	○	○	○
🟨	Case Execution Engine	◐	○	◐	○	○	◐	○	◐	○	◐	●	●	●	○	◐
🟪	Case-Based Process Models	◐	○	◐	○	○	◐	○	◐	○	◐	○	○	○	●	○
🟩	Higher-Order Functional Language	◐	◐	◐	○	○	○	○	○	○	○	○	○	○	○	○
🟧	Advanced Search & Indexing	◐	○	○	○	○	○	○	○	○	○	○	○	○	○	○
🟥	Role-Based & Discretionary Access Control Models	◐	○	○	●	○	◐	●	○	○	◐	○	○	○	○	◐
🟦	Multiple Dynamic Schemata	◐	◐	○	○	○	◐	○	○	◐	○	○	○	○	○	○
🟦	Annotated Versioned Linked Content Graph	◐	○	○	○	○	◐	○	○	◐	○	○	○	○	○	○
	$\sum$	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

○ not supported ◐ partly supported ● fully supported

Table 4.4.: Summary of the conceptually supported requirements.

CHAPTER 5

Prototypical Implementation

This chapter focuses on prototypical implementation. First, crucial end-user interface features are explained based on screens showing sample models in Section 5.1. The model definition reference, including sample declarations, is described in Section 5.2. Modeling case templates is a complex endeavor. Therefore, we summarized the best practice principles we repeatedly noticed within several use cases in Section 5.3. To bridge the gap from a declared case template to an executable model, the import workflow, including related dependency resolution, is described in Section 5.4. Executable cases must be visually accessible for end-users and technically accessible for external systems. Therefore, the conceptual API design is described in Section 5.5. Several fundamental technical challenges are illustrated in Section 5.6. Finally, the supported requirements are summarized in Section 5.7.

5.1. End-User Interface Features

This section illustrates the end-user interface features[1]. The user interface is primarily designed to be used as a web-based desktop application but supports responsive design where possible to allow occasional access with a tablet or smartphone. All features were tested with Google Chrome and Microsoft Edge, but other browsers usually work as well.

All views support switching the localization and related language to English (EN), Spanish (ES), Catalan (CA), Dutch (NL), Russian (RU) and Hebrew (IL). While for the first five languages, the layout is aligned from Left To Right (LTR), the layout is aligned from Right To Left (RTL) for Hebrew. A purely meta-model-based system requires that localizations, such as name translations, are contained within the meta-model to provide a single point of truth. Currently, all meta-model elements support the declaration of one language, which implies that meta-model elements do not switch the language automatically when changing the language on the client application.

[1] The end-user interface was implemented in cooperation with the Advanced Digital Innovation, Salts Mill, Victoria Road, Saltaire, West Yorkshire BD18 3LA, www.adi-uk.com.

In the following, the end-user interface features are explained in detail. First, the single-sign-on and multi-tenancy support is illustrated in Section 5.1.1. After a user successfully signed in, the dashboard is shown by default, as presented in Section 5.1.2. The my-case view shows all accessible cases for the currently signed-in user, as is illustrated in Section 5.1.3. A case detail view can be opened either from dashboard elements linking to a specific case or from the my-case view, which is elaborated in Section 5.1.4 - 5.1.14. Users and roles can be managed centrally, as presented in Section 5.1.15.

All screens presented in the following were taken from our test system with the same build version as the production system and showing dummy data which is as close as possible to the production data to comply with data privacy regulations.

5.1.1. Single Sign-On and Multi-tenancy

Providing a scalable web application requires the management of multiple cases from different organizations with one physical deployment. Data privacy regulations enforce an explicit separation of organizational data which leads to a multi-tenancy architecture. The user interface supports this with the concept of workspaces that represents tenants. The access rights of the users are modeled so they have access to one or multiple workspaces.

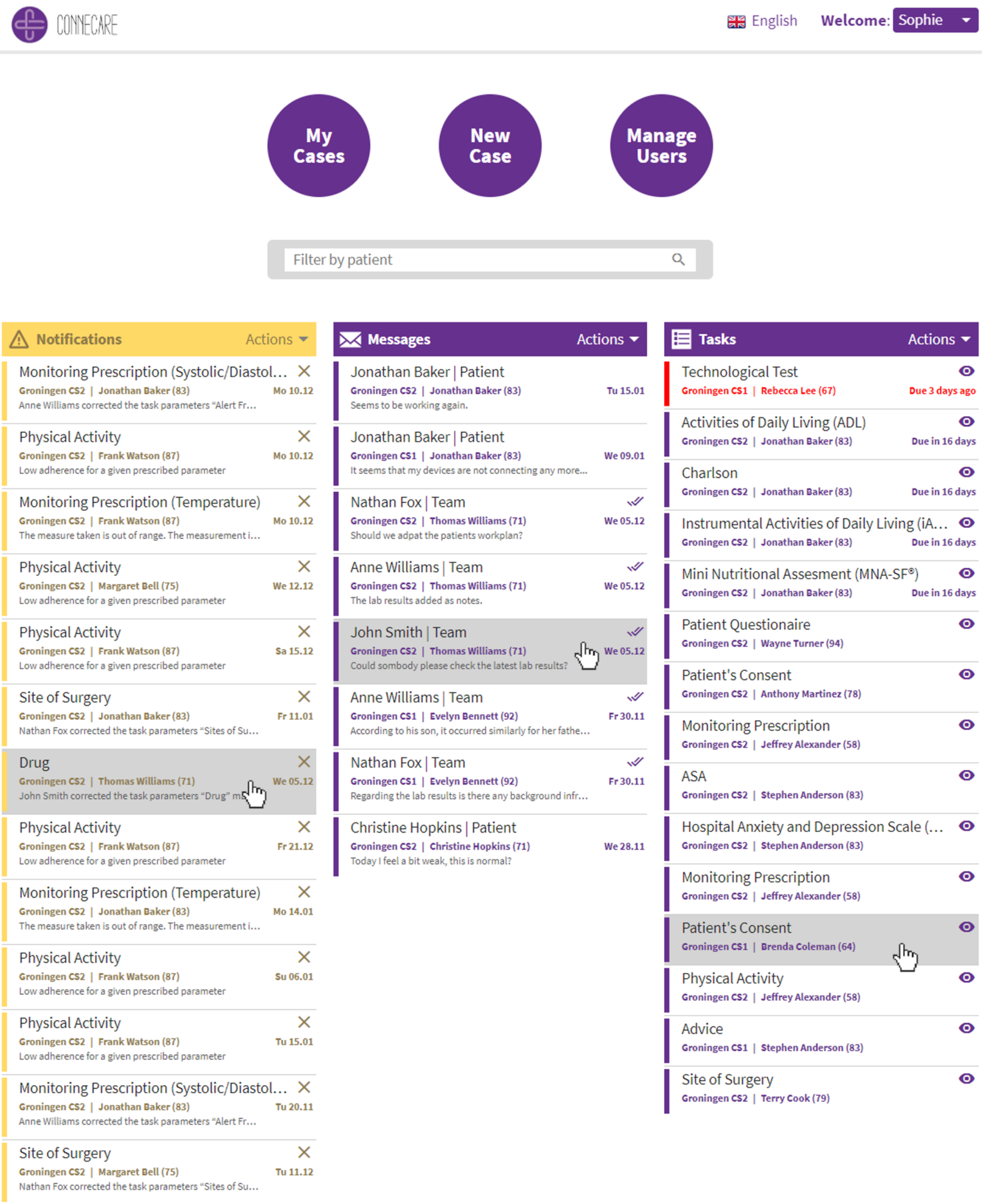
A Single Sign-On (SSO) strategy is supported to enable integrating the ACM4IC approach as a service into an integrated care architecture. Additionally, the SSO enables seamless integration of the patient messaging service (cf. Section 5.1.13) that is not part of the ACM4IC approach. A user signs in with his username or email, password, and the desired tenant that represents a workspace in the meta-model. If the user does not have sufficient access rights for the selected workspace, the login is denied. Technically, the workspace could be selected after the SSO, but due to usability considerations, this step is integrated into the login procedure. All requests use the secured HTTPS protocol for data transmission.

5.1.2. Dashboard

The dashboard provides a user-specific overview to quickly identify needed contrition. All information and actions shown depend on the user's access rights and roles. After a successful login, the user is automatically forwarded to the dashboard that shows navigation options and provides an overview of notifications, messages, and tasks as illustrated in Figure 5.1.

On the first third of the screen shows the navigation options to open the my-case view (cf. Section 5.1.3), to open the create new case page, and to open the management of users and roles view (cf. Section 5.1.15). New cases can only be instantiated by users that have a role compliant with the case definition owner constraint.

On the lower two thirds of the screen are structured into three columns showing the non-acknowledged notifications, unread messages, and pending tasks of the currently signed-in user. Above, a filter option allows to filter all three columns simultaneously according to a patient.



Notifications

Monitoring Prescription (Systolic/Diastol…
Groningen CS2 | Jonathan Baker (83) Mo 10.12
Anne Williams corrected the task parameters “Alert Fr…

Physical Activity
Groningen CS2 | Frank Watson (87) Mo 10.12
Low adherence for a given prescribed parameter

Monitoring Prescription (Temperature)
Groningen CS2 | Frank Watson (87) Mo 10.12
The measure taken is out of range. The measurement i…

Physical Activity
Groningen CS2 | Margaret Bell (75) We 12.12
Low adherence for a given prescribed parameter

Physical Activity
Groningen CS2 | Frank Watson (87) Sa 15.12
Low adherence for a given prescribed parameter

Site of Surgery
Groningen CS2 | Jonathan Baker (83) Fr 11.01
Nathan Fox corrected the task parameters “Sites of Su…

Drug
Groningen CS2 | Thomas Williams (71) We 05.12
John Smith corrected the task parameters “Drug” m

Physical Activity
Groningen CS2 | Frank Watson (87) Fr 21.12
Low adherence for a given prescribed parameter

Monitoring Prescription (Temperature)
Groningen CS2 | Jonathan Baker (83) Mo 14.01
The measure taken is out of range. The measurement i…

Physical Activity
Groningen CS2 | Frank Watson (87) Su 06.01
Low adherence for a given prescribed parameter

Physical Activity
Groningen CS2 | Frank Watson (87) Tu 15.01
Low adherence for a given prescribed parameter

Monitoring Prescription (Systolic/Diastol…
Groningen CS2 | Jonathan Baker (83) Tu 20.11
Anne Williams corrected the task parameters “Alert Fr…

Site of Surgery
Groningen CS2 | Margaret Bell (75) Tu 11.12
Nathan Fox corrected the task parameters “Sites of Su…

Messages

Jonathan Baker | Patient
Groningen CS2 | Jonathan Baker (83) Tu 15.01
Seems to be working again.

Jonathan Baker | Patient
Groningen CS1 | Jonathan Baker (83) We 09.01
It seems that my devices are not connecting any more...

Nathan Fox | Team ✓✓
Groningen CS2 | Thomas Williams (71) We 05.12
Should we adpat the patients workplan?

Anne Williams | Team ✓✓
Groningen CS2 | Thomas Williams (71) We 05.12
The lab results added as notes.

John Smith | Team ✓✓
Groningen CS2 | Thomas Williams (71) We 05.12
Could sombody please check the latest lab results?

Anne Williams | Team ✓✓
Groningen CS1 | Evelyn Bennett (92) Fr 30.11
According to his son, it occurred similarly for her fathe…

Nathan Fox | Team ✓✓
Groningen CS1 | Evelyn Bennett (92) Fr 30.11
Regarding the lab results is there any background infr…

Christine Hopkins | Patient
Groningen CS2 | Christine Hopkins (71) We 28.11
Today I feel a bit weak, this is normal?

Tasks

Technological Test
Groningen CS1 | Rebecca Lee (67) Due 3 days ago

Activities of Daily Living (ADL)
Groningen CS2 | Jonathan Baker (83) Due in 16 days

Charlson
Groningen CS2 | Jonathan Baker (83) Due in 16 days

Instrumental Activities of Daily Living (iA…
Groningen CS2 | Jonathan Baker (83) Due in 16 days

Mini Nutritional Assesment (MNA-SF®)
Groningen CS2 | Jonathan Baker (83) Due in 16 days

Patient Questionaire
Groningen CS2 | Wayne Turner (94)

Patient's Consent
Groningen CS2 | Anthony Martinez (78)

Monitoring Prescription
Groningen CS2 | Jeffrey Alexander (58)

ASA
Groningen CS2 | Stephen Anderson (83)

Hospital Anxiety and Depression Scale (…
Groningen CS2 | Stephen Anderson (83)

Monitoring Prescription
Groningen CS2 | Jeffrey Alexander (58)

Patient's Consent
Groningen CS1 | Brenda Coleman (64)

Physical Activity
Groningen CS2 | Jeffrey Alexander (58)

Advice
Groningen CS1 | Stephen Anderson (83)

Site of Surgery
Groningen CS2 | Terry Cook (79)

Figure 5.1.: Dashboard page showing navigation options and needed contributions.

On the left, the yellow notifications column lists all non-acknowledged notifications that belong to tasks which are owned by the user. Notifications can represent critical information which needs immediate action. Therefore, the case owner's dashboard contains the notification as well. The notification preview shows the related task title in the first line, the case definition name, the patient including their age, and the date when the notification occurred in the second line. In the third line, the notification text is shown as a preview. Clicking on a notification opens the task representation (cf. Section 5.1.8) and shows the notification details on top. Notifications that need no interaction can be acknowledged directly on the dashboard with the acknowledge button on the top right corner and the notification is removed from the dashboard afterwards. Additionally, all listed notifications can be acknowledged simultaneously with an action in the context menu.

In the center, the message column lists all unseen messages that are either from the case team or directly from the patient. Each listed message preview shows the author and indicates whether it is a team or patient message in the first line, as well as the case definition name, the patient including the age, and the messages send date in the second line. Below, a one-line preview present shows the message content. Clicking on a message opens the related conversation in the case and shows the full message (cf. Section 5.1.13). All team-related messages can quickly be marked as seen on the top right corner and then disappear on the dashboard. Similar to the notifications, all listed messages can be marked as seen simultaneously with the context menu actions.

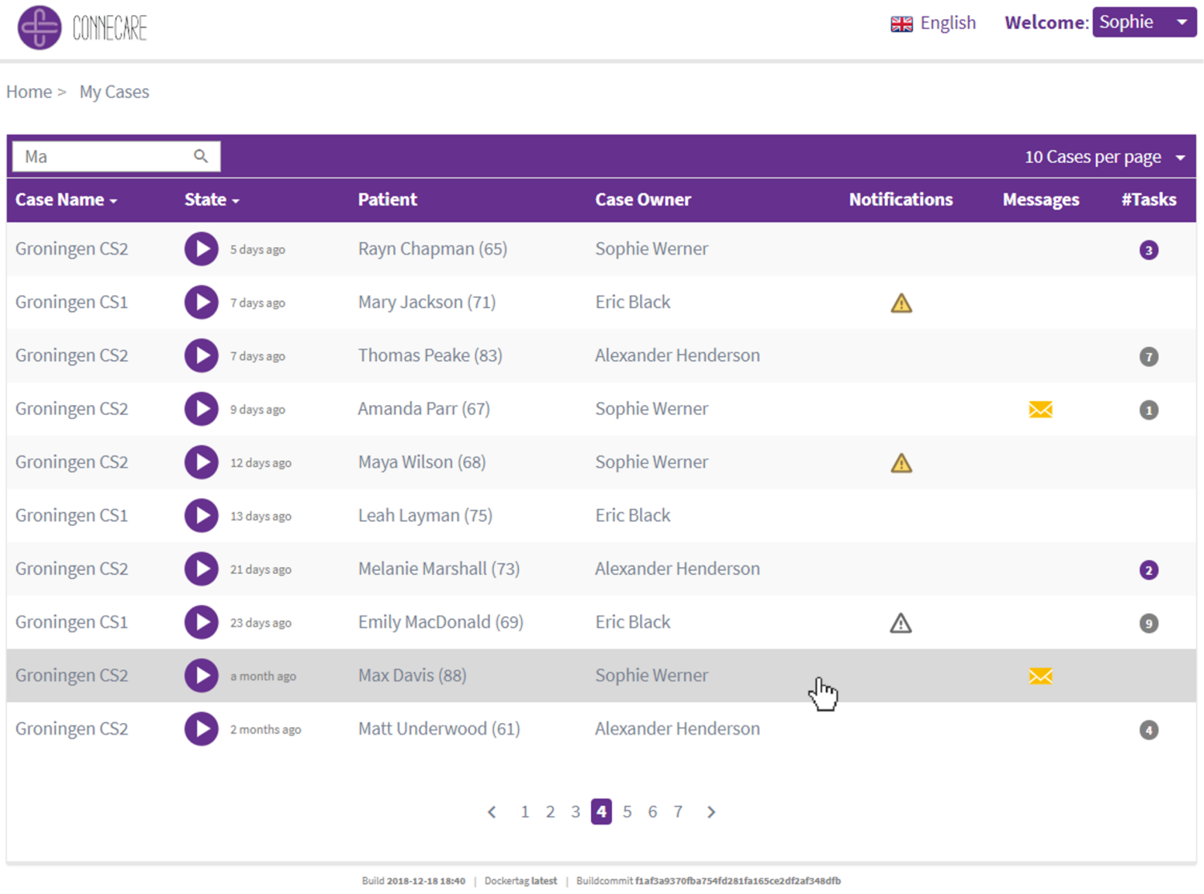
On the right, the task column lists all currently active tasks where the current user is assigned as an owner. The tasks are listed according to priority: due tasks come first and are indicated in red, followed by tasks that have an assigned due date and then tasks without a set due date. Each task shows the task title in the first row and in the second row, it displays the related case definition name, the patient including the age, and the due date if applicable. Clicking on a task opens the related detailed task representation (cf. Section 5.1.8). Users are empowered to hide active tasks from their dashboard to better manage their active tasks. Clicking on the hide button which is visible at the task's top right corner lets a task disappear. The context menu allows to show all hidden tasks and to unhide them. Additionally, all tasks can be hidden or unhidden at once.

5.1.3. My-Cases

The my-case page lists all cases depending on the access rights of the currently logged in user, as illustrated in Figure 5.2. The primary purpose is either to navigate to a dedicated patient case quickly or to identify cases where a contribution is needed.

All accessible cases are represented as a table with the following information: the case definition name that allows identifying the case template, the current case state that is visualized with a state icon in combination with the passed time since the last state transition was performed, the patient including the age that is assigned to the case, the first and last name of the case owner, case notifications that are currently non-acknowledged; personal notifications are highlighted with a yellow warn icon and notifications assigned to other users are shown with a gray warn icon, unread messages are visualized with a yellow envelope, and a counter visualizes the number of pending tasks. The user's pending tasks are colored in purple and pending tasks of other users are colored in gray. Hovering over an icon shows additional information as a tooltip. The case state tooltip shows the transition name, the first and last name of the user who triggered the transition and the transition date. Additional information, such as the number of personally non-acknowledged notifications and the total number of non-acknowledged notifications per case is provided by the notification tooltip. The task tooltip follows a similar structure, whereas the message tooltip simply shows the number of personally unread messages.

After a state transition, a human contribution is needed most likely. Therefore, the default order lists cases according to the last state transition to identify needed contributions quickly. The second ordering criterion is the case definition name. In practice, the number of cases accessible for a user grows over time. Therefore, a search provides quick access to specific cases. The search allows filtering the results according to the patient id, the patient's name, the patient's email address, and the case id. The number of visible cases needs to be limited to ensure sufficient usability. A pagination feature on the bottom allows to navigate across the results pages, either relative to the next or previous page, or directly to the listed page. For reason of usability, a maximum of five previous and five following pages are shown for the direct navigation. Depending on the user's environment, or use case, the results per page can be changed on the top right corner. Available options are 10, 20, 50, 100, 200, and 500 cases per page.



Case Name ▾	State ▾	Patient	Case Owner	Notifications	Messages	#Tasks
Groningen CS2	▶ 5 days ago	Rayn Chapman (65)	Sophie Werner			3
Groningen CS1	▶ 7 days ago	Mary Jackson (71)	Eric Black	⚠		
Groningen CS2	▶ 7 days ago	Thomas Peake (83)	Alexander Henderson			7
Groningen CS2	▶ 9 days ago	Amanda Parr (67)	Sophie Werner		✉	1
Groningen CS2	▶ 12 days ago	Maya Wilson (68)	Sophie Werner	⚠		
Groningen CS1	▶ 13 days ago	Leah Layman (75)	Eric Black			
Groningen CS2	▶ 21 days ago	Melanie Marshall (73)	Alexander Henderson			2
Groningen CS1	▶ 23 days ago	Emily MacDonald (69)	Eric Black	⚠		9
Groningen CS2	▶ a month ago	Max Davis (88)	Sophie Werner		✉	
Groningen CS2	▶ 2 months ago	Matt Underwood (61)	Alexander Henderson			4

Build 2018-12-18 18:40 | Dockertag latest | Buildcommit f1af3a9370fba754fd281fa165ce2df2af348dfb

Figure 5.2.: The my-cases representation lists all accessible cases.

5.1.4. Case Representation

In the medical context, a case represented one patient treatment such as knee surgery, as illustrated in Figure 5.3. At the top, a breadcrumb shows the current path. Below the actual case is represented with a gray background. Every case has a header summarizing the most important case information. On the left, the current case state is visualized with an icon. Hovering over the icon shows the date and the user's first and last name who triggered the last state transition. To the right of it, the case definition name is represented and when hovering over it, the related case definition version is shown. Relevant patient information, such as the patient first and last name, the patient's age, the currently selected stage, and the unique case identifier is shown. Global case-based actions, such as completing a case, terminating a case, or deleting a case are available for case owner via the drop-down menu placed in the top right corner.

Below, the following case-specific views are represented: the case summary that provides a model-based summary for the instantiated case (cf. Section 5.1.5), the case workflow that contains all process elements, such as `HumanTasks`, `AutomatedTasks`, and `DualTasks` (cf. Section 5.1.6), the data page that presents the linked data generated by the workflow (cf. Section 5.1.10), the team page that allows modifying the team members' access rights and roles (cf. Section 5.1.11), the notifications page that lists all non-acknowledged and acknowledged case notifications (cf. Section 5.1.12), the messages that allow communication within the case team or with the patient (cf. Section 5.1.13), and the case notes that provide the possibility for unstructured documentation (cf. Section 5.1.14).

Beneath the represented case, a footer is showing the build date. A click on the footer toggles between the basic and extended build information that additionally contain the docker tag and the commit hash. The footer is available on most pages to allow reporting issues including the build version.

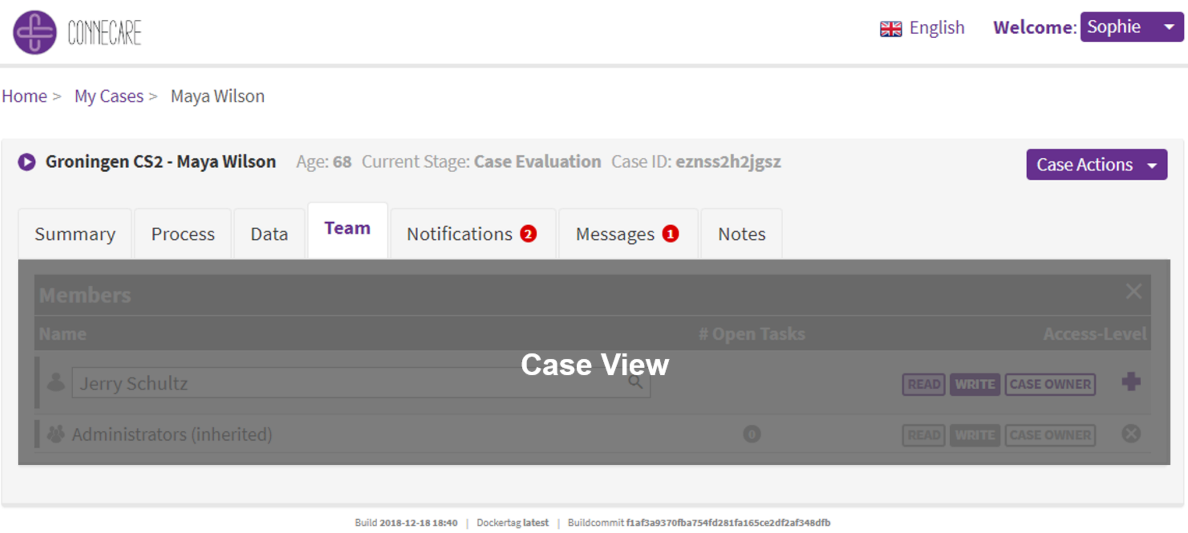
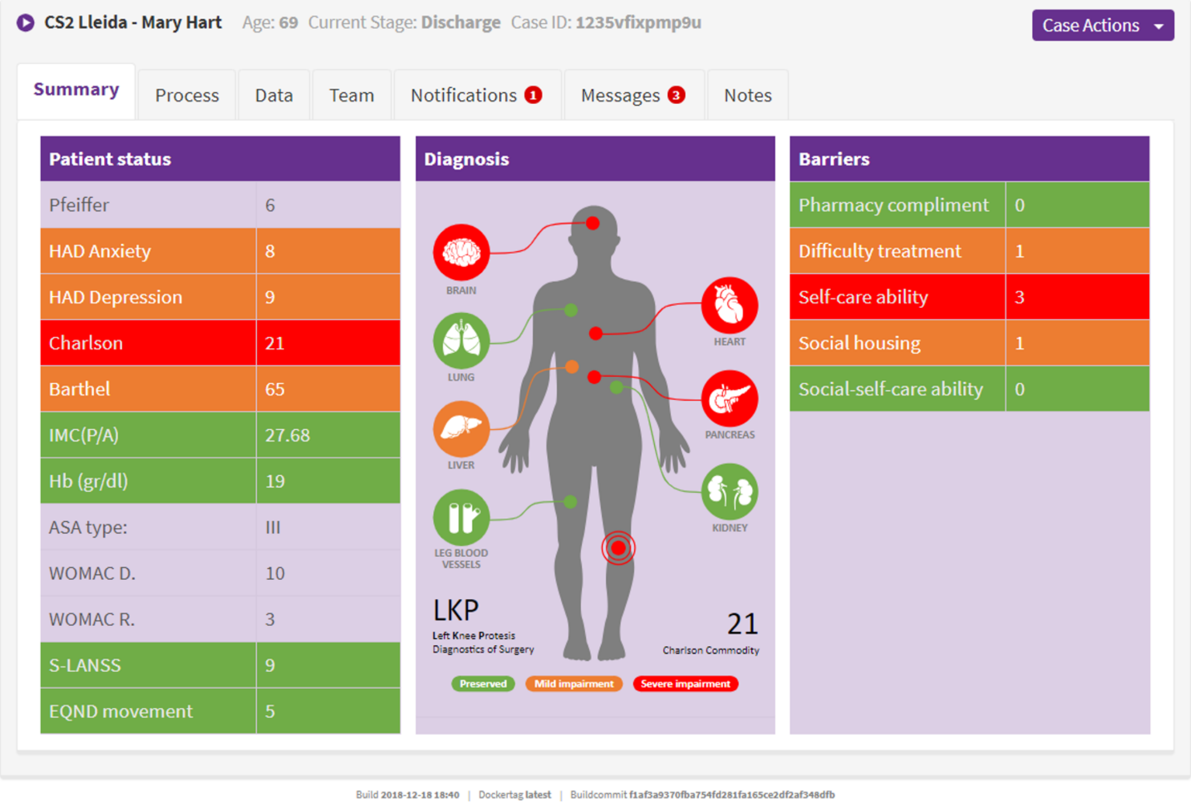

CONNECARE
English
Welcome: Sophie
Home > My Cases > Maya Wilson
Groningen CS2 - Maya Wilson
Age: 68 Current Stage: Case Evaluation Case ID: eznss2h2jgsz
Case Actions
Summary
Process
Data
Team
Notifications 2
Messages 1
Notes
Members
Name
# Open Tasks
Access-Level
Case View
Jerry Schultz
READ WRITE CASE OWNER
Administrators (inherited)
0
READ WRITE CASE OWNER
Build 2018-12-18 18:40 | Dockertag latest | Buildcommit f1af3a9370fba754fd281fa165ce2df2af348dfb

Figure 5.3.: Case representation showing the case header and the case view options.

5.1.5. Case Summary

During a patient visit, care professionals need a quick overview on the patient. Depending on the treatment, different clinical parameters are relevant. Therefore, the case template allows the definition of a summary page that shows data generated by accomplished tasks. Figure 5.4 illustrates a sample summary page including a graphical body representation as used in the clinical environment. A summary page is structured with summary sections that are represented with a white-colored title on a purple background. Below the title, the resolved paths are listed as linked attributes. The linked attributes mostly consist of dynamically calculated scores that summarize a task. Each linked attribute applies the custom data representation defined on the attribute definition, i.e., the coloring or the SVG-based rendering. Attribute values linking to uncompleted tasks with write access for that attribute are hidden until the related tasks are completed to prevent decisions on drafted data.

Providing a visual attractive summary requires custom positioning of summary sections. A grid like a layout with three columns allows positioning each summary section either on the left, in the center, on the right, or stretched over the full width. One column can contain multiple summary section definitions and the last container of the last summary section is vertically stretched to be visually aligned with the other two columns. Clicking on a summary section parameter opens the task detail representation of the task that currently modifies this attribute or, if no task is active any longer, of the task that previously modified the attribute.



CS2 Lleida - Mary Hart Age: 69 Current Stage: Discharge Case ID: 1235vfixpmp9u

Patient status	
Pfeiffer	6
HAD Anxiety	8
HAD Depression	9
Charlson	21
Barthel	65
IMC(P/A)	27.68
Hb (gr/dl)	19
ASA type:	III
WOMAC D.	10
WOMAC R.	3
S-LANSS	9
EQND movement	5

Diagnosis

LKP
Left Knee Protesis
Diagnostics of Surgery

21
Charlson Commodity

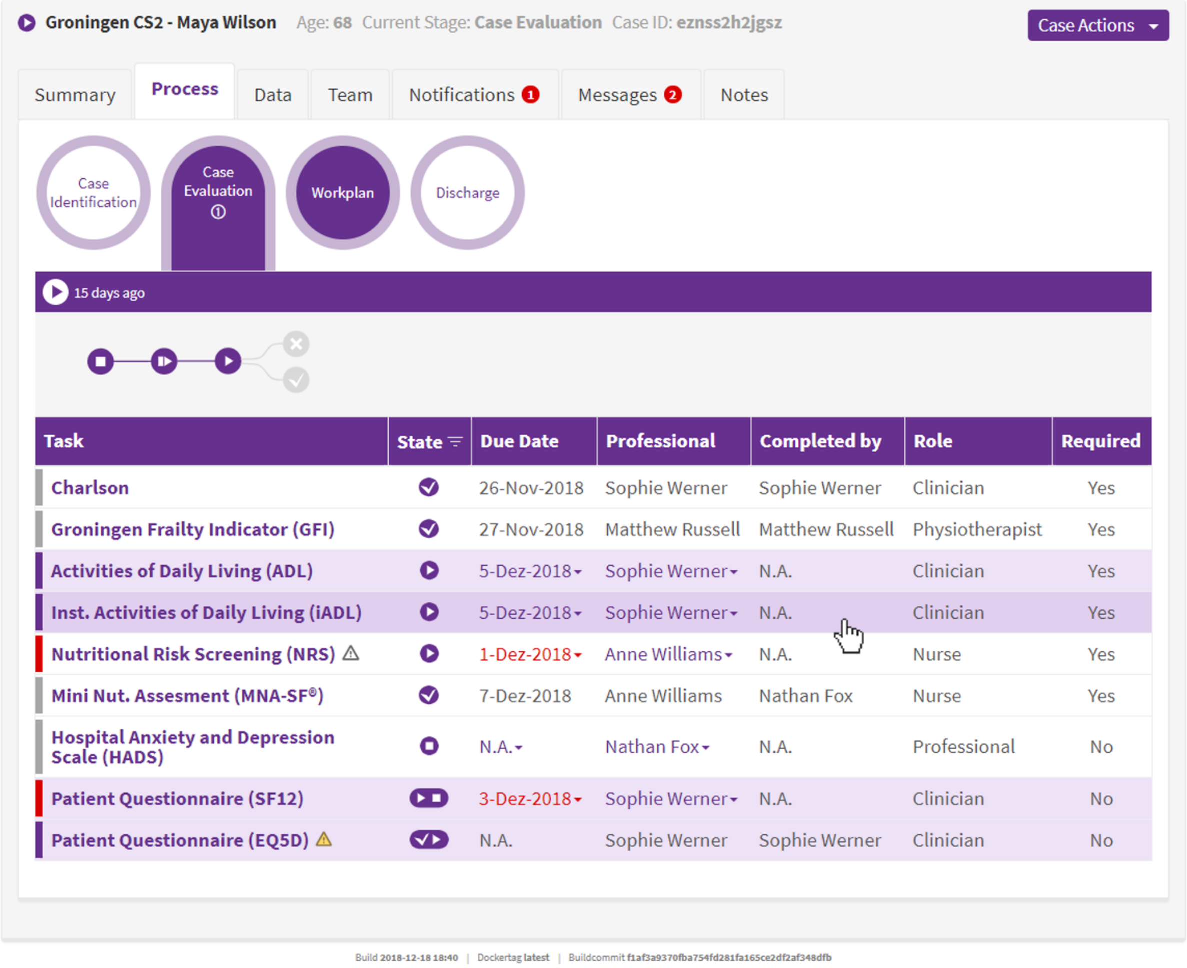
Barriers	
Pharmacy compliment	0
Difficulty treatment	1
Self-care ability	3
Social housing	1
Social-self-care ability	0

Build 2018-12-18 18:40 | Dockertag latest | Buildcommit f1af3a9370fba754fd281fa165ce2df2af348dfb

Figure 5.4.: Case summary page providing a visual attracting overview.

5.1.6. Case Workflow

The case workflow represents the overall patient treatment plan and is primarily structured into stages and related tasks, as illustrated in Figure 5.5. All stages are represented as circles on the top. currently active stages are colored in purple and the currently opened stage is attached to the task table below. Repeatable stages indicate the current iteration with a number below the stage name. The purple header below the stages shows stage details, such as the current state on the left and the possible case actions menu on the right. Stage actions are powerful and may affect all stage subprocesses. Therefore, only case owners can perform those. A stage can be completed successfully at any time if all relates subprocesses are completed or terminated. Clicking on the state extends the header and a stage state diagram is visible. When hovering over a state, the state transition, the user who triggered the state transition, and the related date is shown.



Task	State	Due Date	Professional	Completed by	Role	Required
Charlson	✓	26-Nov-2018	Sophie Werner	Sophie Werner	Clinician	Yes
Groningen Frailty Indicator (GFI)	✓	27-Nov-2018	Matthew Russell	Matthew Russell	Physiotherapist	Yes
Activities of Daily Living (ADL)	▶	5-Dez-2018▾	Sophie Werner▾	N.A.	Clinician	Yes
Inst. Activities of Daily Living (iADL)	▶	5-Dez-2018▾	Sophie Werner▾	N.A.	Clinician	Yes
Nutritional Risk Screening (NRS) ⚠	▶	1-Dez-2018▾	Anne Williams▾	N.A.	Nurse	Yes
Mini Nut. Assesment (MNA-SF®)	✓	7-Dez-2018	Anne Williams	Nathan Fox	Nurse	Yes
Hospital Anxiety and Depression Scale (HADS)	■	N.A.▾	Nathan Fox▾	N.A.	Professional	No
Patient Questionnaire (SF12)	■▶	3-Dez-2018▾	Sophie Werner▾	N.A.	Clinician	No
Patient Questionnaire (EQ5D) ⚠	✓▶	N.A.	Sophie Werner	Sophie Werner	Clinician	No

Build 2018-12-18 18:40 | Dockertag latest | Buildcommit f1af3a9370fba754fd281fa165ce2df2af348dfb

Figure 5.5.: The case workflow shows the current state of the case.

Below the selected stage, a table lists all related tasks, with one row representing a single task. All tasks show information such as a name, a state, an optional due date, an optional task owner, the user who completed the task, the tasks owner role constraint, and a flag indicating whether the task is mandatory. To distinguish repeatable tasks, the models allow the dynamic generation of task name postfixes depending on the task attribute values. The task's state is visualized with a symbol. Human and automated tasks are represented with one state symbol, and dual tasks are represented with two state symbols. The first state of a dual task represents the human part and the second the automated part. In-place editing, indicated with gray arrows, allows to quickly change the due date or the owner without opening the task detail representation. Until the task is completed or terminated, the due date and owner can be changed. For dual tasks, the human part is considered.

Color coding is applied to quickly identify the needed contributions. Overdue tasks are highlighted in red. Acknowledged notifications are indicated with a gray warning icon and non-acknowledged notifications are indicated with a yellow warning icon. All tasks to which the currently signed-in user should contribute are highlighted with a purple background. An active contribution is needed when a task is active or has a non-acknowledged notification.

In practice, the number of stage tasks can be increasing continuously during the case execution due to repeating tasks. Therefore, the user interface allows filtering the stage task list according to the state. Typically, only active tasks are needed to work efficiently.

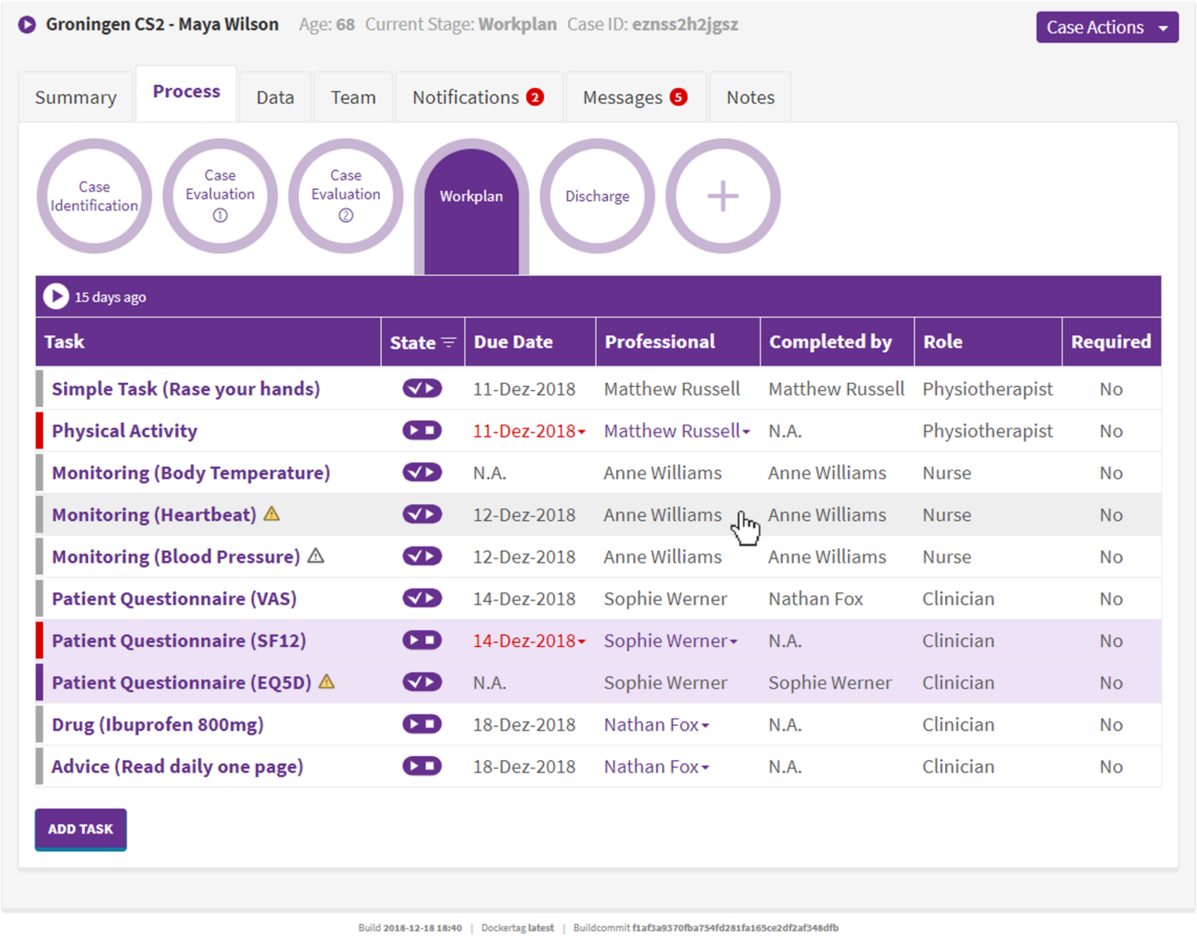
5.1.7. Flexible Process Adaptation during Run-Time

Knowledge-intensive processes are hardly foreseeable upfront. In the medical domain, the patient treatment steps entirely depend on the contextual situation. Defining all possible combinations of treatment steps would lead to a very complex and unusable process model. Therefore, the process model definition allows the predefinition of process fragments that can be assembled to an individual treatment plan.

Figure 5.6 shows an active stage with manual activation tasks. After the task is in the enabled state, a user needs to add those predefined tasks manually to the stage for them to be executed. The add task button is dynamically shown when a task can be activated manually. A dialog lists all tasks available for manual activation. Depending on whether a task is repeatable, multiple instances might be added to a stage. The sequential repetition only allows adding the same task if the previous task is already completed. The parallel repetition allows adding as many tasks desired.

The manual stage activation follows a similar approach. If a stage can be activated manually, the list of stages contains a stage with a plus symbol as the last item. Clicking on the add stage button opens a dialog where all stages for manual activation are listed. Manually activating a stage may also imply updates of the nested process elements. In practice, adding a stage normally also activates at least one subprocess element, such as a task. In the clinical context, the treatment must be validated after a certain time. Therefore, an additional state evaluation is added to compare the initial patient evaluation results from the patient with the evaluation results during or after the individual treatment. An evaluation contains many different human tasks that define clinical health status.

A process may contain tasks or stages that are not suitable for the dedicated case. Those stages or tasks could be terminated by knowledge workers to avoid their execution, whereas the knowledge worker who made the decision to terminate a process needs to ensure that the required tasks and stages still meet the preconditions to be able to continue working on the case.


Groningen CS2 - Maya Wilson
Age: 68 Current Stage: Workplan Case ID: eznss2h2jgsz
Case Actions
Summary
Process
Data
Team
Notifications 2
Messages 5
Notes
Case Identification
Case Evaluation ①
Case Evaluation ②
Workplan
Discharge
+
15 days ago
ADD TASK
Build 2018-12-18 18:40 | Dockertag latest | Buildcommit f1af3a9370fba754fd281fa165ce2df2af348dfb

Task	State	Due Date	Professional	Completed by	Role	Required
Simple Task (Rase your hands)	✓ ▶	11-Dez-2018	Matthew Russell	Matthew Russell	Physiotherapist	No
Physical Activity	▶ ■	11-Dez-2018	Matthew Russell	N.A.	Physiotherapist	No
Monitoring (Body Temperature)	✓ ▶	N.A.	Anne Williams	Anne Williams	Nurse	No
Monitoring (Heartbeat) ⚠	✓ ▶	12-Dez-2018	Anne Williams	Anne Williams	Nurse	No
Monitoring (Blood Pressure) ⚠	✓ ▶	12-Dez-2018	Anne Williams	Anne Williams	Nurse	No
Patient Questionnaire (VAS)	✓ ▶	14-Dez-2018	Sophie Werner	Nathan Fox	Clinician	No
Patient Questionnaire (SF12)	▶ ■	14-Dez-2018	Sophie Werner	N.A.	Clinician	No
Patient Questionnaire (EQ5D) ⚠	✓ ▶	N.A.	Sophie Werner	Sophie Werner	Clinician	No
Drug (Ibuprofen 800mg)	▶ ■	18-Dez-2018	Nathan Fox	N.A.	Clinician	No
Advice (Read daily one page)	▶ ■	18-Dez-2018	Nathan Fox	N.A.	Clinician	No

Figure 5.6.: Flexible process adaptation shows a patient-centered treatment plan manually composed during the case execution.

5.1.8. Task Representation

In the medical domain, questionnaires are used to capture knowledge regarding the patient health status. These questionnaires consist of several ordered questions with mostly different answer options that result in a final questionnaire score to assess the patient. There are common questionnaires, such as the Charlson Comorbidity Index (Charlson et al., 1987) and hospital-specific questionnaires specially developed for a certain local treatment.

The medical questionnaires are modeled as `HumanTasks` or as `DualTasks`, depending on whether the clinical professional or the patient himself fills in the questionnaire via mobile app. Figure 5.7 shows an extended Charlson Comorbidity questionnaire modeled as a `HumanTask`. Figure 5.8 illustrates a patient's long-term monitoring task modeled as a `DualTask` where first the care professional defines the boundary parameters, and subsequently, the patient performs the defined measurement within the defined time slot and measurement frequency.

All task pages follow a similar structure, regardless of the actual task type (`HumanTask`, `DualTask`, or `AutomatedTask`). A breadcrumb above the opened case shows the path to the currently active task page, including the containing stage. A task title allows associating the content of the task on the stage's task-list as well (cf. Figure 5.5). Below the title, possibly occurring notifications are visible. A gray block shows the most relevant task meta-data followed by the task's content. The task controls are placed below. On the bottom, a gray colored footnote shows copyrights and references if needed.

Each task has several meta-data fields to track the task's current state and to enable collaborative working within a case. A task owner field allows defining a user who is responsible for accomplishing a task while considering that the assigned user needs to fulfill the role constraint. Typically, when a task becomes active, the task's owner is already predefined through a previous task that assigned users to all roles of a case. All tasks that need human interaction (`HumanTasks` and `DualTasks`) have a due date field that indicates until when the task should be accomplished. When the task becomes overdue, the due date is highlighted in red. Until a task is completed or terminated, the default task owner and default due date can be changed. Changing an owner or due date does not modify the linked attribute values, because those are is mostly used for multiple tasks. Instead, the values are overridden locally. Global role or due date changes are possible on the team page or on the task page where the linked attributes representing the owner and due date can be modified. Each task follows a lifecycle indicated with a state diagram. Accomplished or active states are colored in purple and non-accomplished states are visualized in gray. Hovering over a state shows when and by which user the state transition was triggered, to ensure traceability. Additionally, the current task state is listed as explicit meta-data field for reason of usability. Conceptually, the `DualTasks` lifecycle represents a `HumanTask` lifecycle followed by an `AutomatedTask` lifecycle which is performed by a patient in another system. The diagram shows that combined lifecycle in a shortened and understandable representation.

Task parameters represent the task's main content depending on the underlying model. A task typically has multiple task parameters which itself refers to an attribute containing the core information. Each visible task parameter follows a similar representation pattern. A task parameter headline determined from the related attribute definition helps to identify the question quickly. If additional explaining information is necessary, it can be provided within the attribute definition and is presented with a purple question mark showing a tooltip when hovering over it. A red star indicates that a task parameter is mandatory to complete a task. The attribute constraint determines the task parameter answer type. To provide a purely model-based approach, each type is bound to a default writable representation: a *string* type to a simple input field, a *longtext* type to a textarea, a *boolean* to a checkbox, a *number* type to an input field showing an increase and decrease arrow while only allowing numeric values, an *enumeration* type to radio inputs if the multiplicity allows only one answer and to checkboxes if the multiplicity allows multiple answers, a *date* type to an input field with an attached date picker, a *link* type to a search input with auto-complete, a *json* type to a simple read-only formatted text representation. The read-only representation is comparably simple and shows the answers as well-formatted strings. Further customization to increase the usability and acceptance is possible with custom data representations (cf. Section 5.1.9) that override the default representation. Figure 5.7 shows a derived attribute of the type string interpreted as an SVG image and Figure 5.8 shows time-series measurements of the type json as multiple line diagrams. Depending on the task's use case, some task parameters may be modeled as read-only, e.g., if previously existing information should be visible but immutable during the task execution.

Task parameters are either linked to a simple attribute with direct input values such as enumerations or a derived attribute that calculates a value based on a defined expression. The model-based expressions are expressive and may reference other entities that the calculation needs to be performed on the backed. Therefore, tasks contain task parameters which are link to derived attributes, drafting all changes to the backend to receive updated derived attributes on the fly. This pattern allows to provide immediate feedback for the user's input. Users can understand the overall impact of a single answer option immediately, which increases usability. Considering the sample illustrated in Figure 5.7, the body representation summarizes all information that is immediately visual. The diagnosis of surgery is highlighted with a red point, the organs are colored according to their condition evaluated within the task and the final Charlson score is provided. Another derived attribute calculates the overall Charlson score and uses the custom data representation to visually indicate with colors whether the resulting score is sufficient or not.

Stages with a high degree of flexibility typically contain manual activation tasks that can be repeated in parallel. In the medical domain, the patient-centric treatment is typically modeled as such a stage (cf. Figure 5.6). Care professionals manually activate a monitoring task and define a blood pressure monitoring, manually activate a monitoring task and define a temperature measurement, and manually activate many more monitoring tasks if needed. Depending on what task parameters are selected, the task provides fundamentally different functionality, but the task name is similar. Having multiple tasks with a similar name leads to confusion on the task stage list. Therefore, the task names can be dynamically adapted depending on the selected task parameters. Each task supports defining one task parameter that is used to dynamically enrich the task title after the task parameter value is selected. Figure 5.8 shows a dynamically enriched title with the value blood pressure in brackets. This pattern allows using abstract task models while ensuring maximum usability.

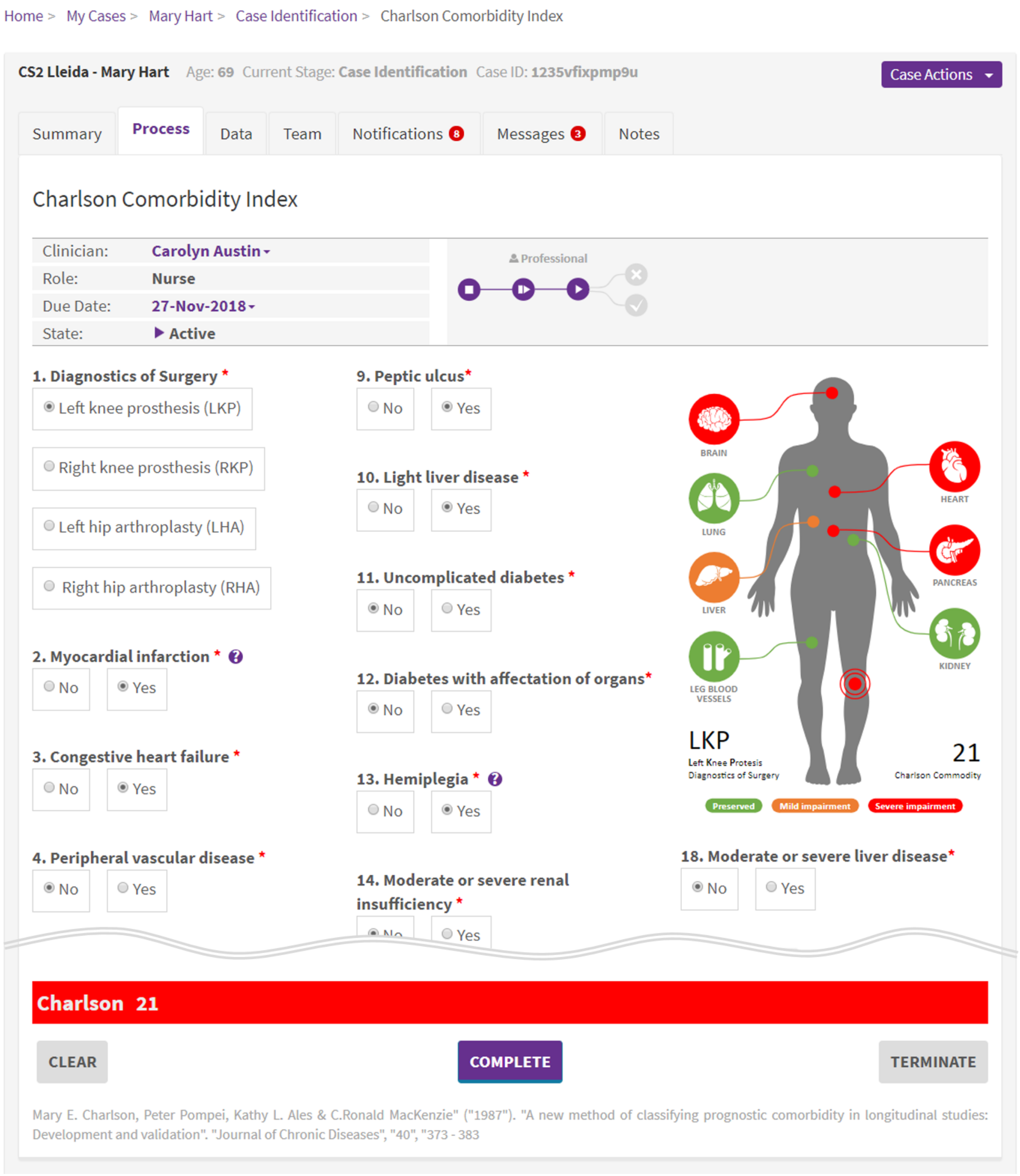
All task pages support several layout options for task parameters. By default, all task parameters are stretched to the full width and listed below each other. In general, the page consists of a grid with three columns that allow positioning parameters on the left, center, right or stretched over all columns. Additionally, task parameters can be stretched over two columns, such as left-center or center-right. The responsive design approach applies the default one column layout when the page is shown below the default resolution.

Depending on the current task's state, different action buttons are provided. Active `HumanTasks` provide a primary action to successfully complete the task after all task parameter constraints are fulfilled, or a secondary action to unsuccessfully terminate the task to stop its execution. Additionally, a clear button resets all task parameters to their initial default values. Values that are set with a previous task may be lost because they cannot be treated as default values. `DualTasks` provide similar actions when the human part is active. When the automated part is active, no primary action is available. A manual termination is still allowed as a secondary action to allow flexible process adaption.

While completing a task, input mistakes may occur. Conceptually, completing or terminating a task leads to an immutable task state. Considering that a relevant patient parameter is documented wrongly, a correction is desirable. Therefore, all completed or terminated tasks that contain a human iteration can be corrected manually and are documented for reason of traceability.

Knowledge-intensive processes such as a patient treatment need to deal with unforeseeable occurring events. While monitoring the blood pressure, the minimum or maximum defined thresholds may be exceed, as illustrated in Figure 5.8. Therefore, each process elements supports attaching different notifications types. A task that contains a notification indicates that with a warning icon behind the task title. A yellow warning icon indicates that there is at least one non-acknowledged notification, and a gray icon indicates there are only already acknowledged notifications. All non-acknowledged notifications are colored yellow and listed in reverse chronological order below the task title. Error notifications are represented with simple text and occur on system failures, i.e., a hook execution cannot reach an external API endpoint. Correct notifications make corrections of completed or terminated tasks traceable. Therefore, the user who corrected the task is named and each corrected task parameter is listed with the old and new values. Custom notifications are used to integrate external domain-specific notifications, such as the patient blood pressure exceeding a maximum threshold. All occurring notifications are shown on the task owner dashboard and can be acknowledged at any time either on the dashboard, on the notifications page or directly on the task page. Clicking on the alert icon next to the title fades in the acknowledged notifications below the current notification in gray. For reason of traceability, each notification indicates when the notification was acknowledged by who.

The assigned task owner is responsible that the task is accomplished and that unforeseen occurring events are managed. However, all task actions, i.e., completing a task, terminating a task, correcting a task, or acknowledging an alert can be performed by any case-writer to enable flexible collaboration. The performed actions are documented and traceable for the whole case team, which prevents the abuse of the provided degree of freedom.



Home > My Cases > Mary Hart > Case Identification > Charlson Comorbidity Index

CS2 Lleida - Mary Hart Age: **69** Current Stage: **Case Identification** Case ID: **1235vfixpmp9u**

Charlson Comorbidity Index

Clinician:	**Carolyn Austin** ▾
Role:	**Nurse**
Due Date:	**27-Nov-2018** ▾
State:	▶ **Active**

1. Diagnostics of Surgery *

● Left knee prosthesis (LKP)

○ Right knee prosthesis (RKP)

○ Left hip arthroplasty (LHA)

○ Right hip arthroplasty (RHA)

2. Myocardial infarction * ?

○ No ● Yes

3. Congestive heart failure *

○ No ● Yes

4. Peripheral vascular disease *

● No ○ Yes

9. Peptic ulcus*

○ No ● Yes

10. Light liver disease *

○ No ● Yes

11. Uncomplicated diabetes *

● No ○ Yes

12. Diabetes with affectation of organs*

● No ○ Yes

13. Hemiplegia * ?

○ No ● Yes

14. Moderate or severe renal insufficiency *

● No ○ Yes

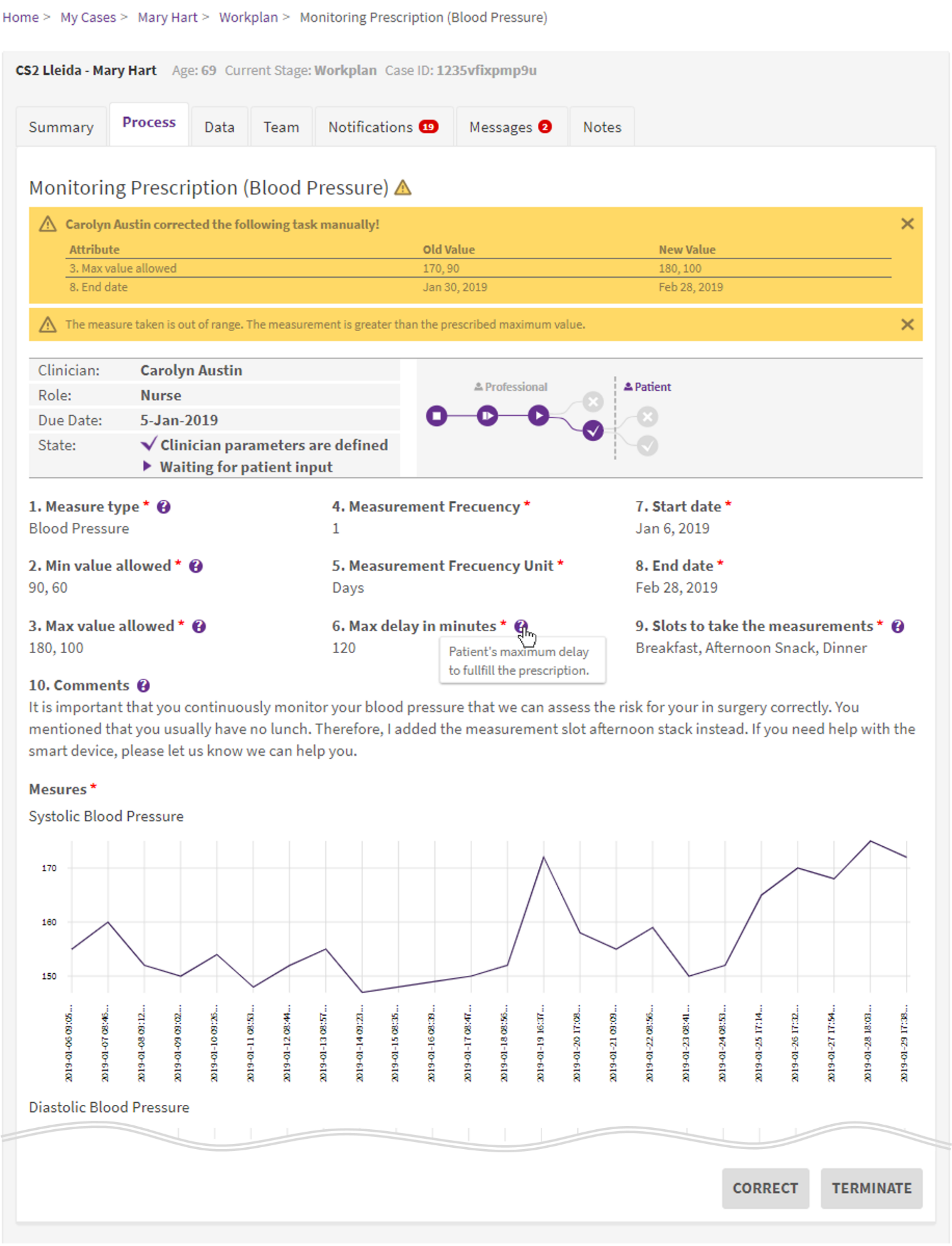
18. Moderate or severe liver disease*

● No ○ Yes

Charlson 21

Mary E. Charlson, Peter Pompei, Kathy L. Ales & C.Ronald MacKenzie" ("1987"). "A new method of classifying prognostic comorbidity in longitudinal studies: Development and validation". "Journal of Chronic Diseases", "40", "373 - 383

Figure 5.7.: `HumanTask` page illustrating a medical questionnaire with a graphical visualization of the results and a dynamically calculated score.



Home > My Cases > Mary Hart > Workplan > Monitoring Prescription (Blood Pressure)

CS2 Lleida - Mary Hart Age: **69** Current Stage: **Workplan** Case ID: **1235vfixpmp9u**

Monitoring Prescription (Blood Pressure) ⚠

⚠ **Carolyn Austin corrected the following task manually!** ✕

Attribute	Old Value	New Value
3. Max value allowed	170, 90	180, 100
8. End date	Jan 30, 2019	Feb 28, 2019

⚠ The measure taken is out of range. The measurement is greater than the prescribed maximum value. ✕

Clinician:	**Carolyn Austin**
Role:	**Nurse**
Due Date:	**5-Jan-2019**
State:	✓ **Clinician parameters are defined** ▶ **Waiting for patient input**

1. Measure type * ?
Blood Pressure

4. Measurement Frecuency *
1

7. Start date *
Jan 6, 2019

2. Min value allowed * ?
90, 60

5. Measurement Frecuency Unit *
Days

8. End date *
Feb 28, 2019

3. Max value allowed * ?
180, 100

6. Max delay in minutes * ?
120

Patient's maximum delay to fullfill the prescription.

9. Slots to take the measurements * ?
Breakfast, Afternoon Snack, Dinner

10. Comments ?
It is important that you continuously monitor your blood pressure that we can assess the risk for your in surgery correctly. You mentioned that you usually have no lunch. Therefore, I added the measurement slot afternoon stack instead. If you need help with the smart device, please let us know we can help you.

Mesures *

Systolic Blood Pressure

Diastolic Blood Pressure

Figure 5.8.: `DualTask` page shows a monitoring prescription defined and monitored by a care professional (human part) and performed by a patient with a third-party system (automated part).

5.1.9. Custom Data Representation

The user interface representation follows the purely meta-model-based approach. This implies that each attribute type is bound to exactly one generic frontend representation. Using a generic attribute-type-specific representation has advantages and disadvantages. One advantage is the simplicity of the models which do not require the specification of any representation, and one disadvantage is the missing flexibility for customization. Most use cases do not need a custom representation. Domain-specific practice-proven representations influence system usability and user acceptance in a positive way. E.g., in the medical context, it is common to highlight critical patient issues on a schematic body representation. Semantically, this information could also be expressed with a generic textual attribute representation, but the usability is not nearly similar as a graphical representation. Therefore, the generic default representations might be overwritten with the following representations:

- **Patient questionnaires** In the context of integrated care, patients are involved in collecting information that supports the medical evaluation. Therefore, it is common that patients need to fill questionnaires on a regular base to detect changes over time, among other things. First, a clinical professional defines the questionnaire meta-data such as i) the type of questionnaire, ii) the start and end date, iii) the measurement frequency, e.g., daily or weekly, and iv) the expected measurement time slot(s) during the day, e.g., during dinner or before sleeping. As a result, the care professional expects a time series of questionnaires. Modeling the described behavior would lead to a task defining the questionnaire meta-data, which then triggers creating multiple patient questionnaire tasks to comply with the defined questionnaire meta-data. Considering that the user interface follows the purely meta-model-based approach, many questionnaires would be visible, which would lead to usability issues for the care professionals. To allow nested questionnaires while having a simplistic user interface to ensure the usability and to support the purely meta-model-based approach, the resulting patient questionnaires are modeled as JSON attribute that is visualized as time series of questionnaires within the task defining the meta-data. The developed meta-model-based visualization concept generically represents the common task parameter types to be reusable. All task parameters are shown in read-only mode. Editing on the user interface is not foreseen, as the data is received via an API endpoint. Figure 5.9 illustrates an EQ5D-5L questionnaire developed by Herdman et al. (2011) as a sample patient questionnaire.

- **Threshold-based coloring** In the clinical context, it is essential to identify critical parameters quickly. Therefore, the user interface supports coloring attribute values depending on thresholds defined within the model. For numeric attribute values, a range can be defined for each color in which the entire set of HTML colors is supported. Coloring attribute values from type string and enumeration are supported as well. The typical use case is to color numerically derived attribute values that summarize a task. Colors are typically defined according to a traffic light schema, i.e., red for a bad value, yellow for an acceptable value, and green for an ideal value. Figure 5.4 shows threshold-based coloring on a summary page, Figure 5.7 illustrates the related coloring on a task page, and Figure 5.10 represents the coloring on a data page.

- **Diagrams for time series data** The expressive power of diagrams allows to identify anomalies in time series data quickly. In the medical context, this is useful to detect inferences while monitoring the patient's heart rate, blood pressure, body temperature, or similar other parameters. Therefore, attribute values of the type JSON can be represented as a series of line diagrams. Each line diagram consists of many data points that have a date and a corresponding numeric value. Figure 5.8 shows a `DualTask` with a JSON attribute represented as a diagram.

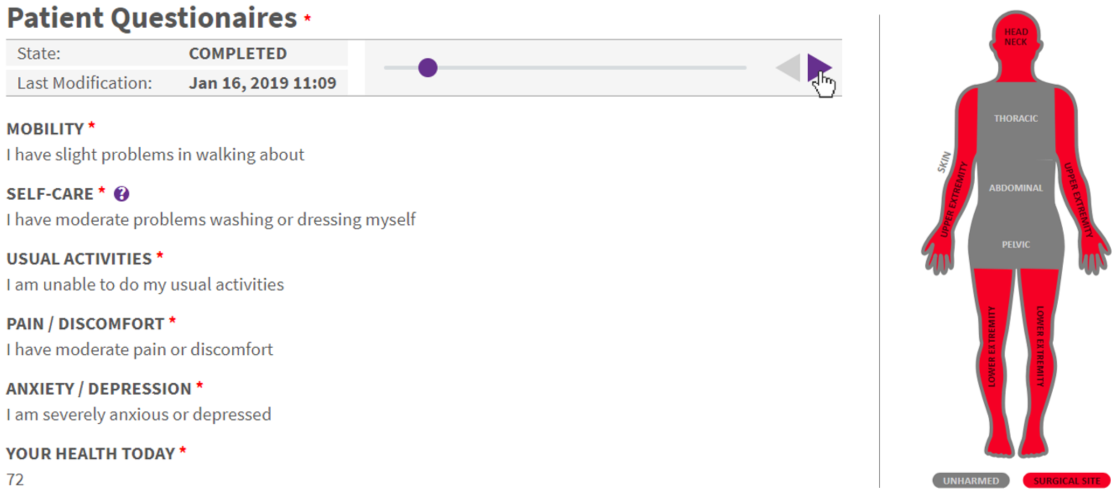
- **Scalable vector graphic representation** In the medical context, schematic human body representations are commonly used to highlight critical patient parameters. The graphical notation helps to identify possibly occurring issues upfront. The user interface supports interpreting an attribute value of the type string as a Scalable Vector Graphic (SVG). The SVG template with placeholders indicating the actual values is modeled as a string attribute that contains the template as a default value. An additional derived attribute calculates the actual values for the placeholders and dynamically builds the SVG as a string. This generic meta-model-based approach can be applied whenever a custom graphical representation is needed. The task representation illustrated in Figure 5.7 graphically visualizes the questionnaire's results with a dynamically calculated SVG. The resulting SVG image is linked on the summary page, as shown in Figure 5.4. An additional sample is represented in Figure 5.9.

- **Conditional Multiplicity** While conceptualizing questionnaires into task definitions, the degree of abstraction is increasing. With a higher degree of abstraction, model elements are more likely to be reusable. The heart rate, body temperature, and systolic/diastolic blood pressure are typical clinical parameters to be monitored. Considering that most medical monitoring tasks are technically structured in a very similar way and could have the following task parameters in common: i) the type of measurement, ii) the start date, iii) the end date, iv) a min threshold, v) a max threshold, and vi) the resulting measurement time series data. The first parameter allows the selection of the measurement type, which is either the heart rate, body temperature, or systolic/diastolic blood pressure. Choosing the start and end date works for all three measurement types in the exact same way. A similar definition of the thresholds for all three measurement types is challenging due to the different multiplicity for the systolic/diastolic blood pressure thresholds. Therefore, a conditional multiplicity allows defining a dynamically adaptable multiplicity depending on the selected value of another task parameter. We Assume that the modeled default multiplicity is within the range of the dynamic adaptation; otherwise, model validation errors will occur. Alternatively, the blood pressure task could be separated, but considering a larger amount of actual measurement types justifies the conditional multiplicity modeling capability. Figure 5.8 contains the example above-described.

- **External Enumerations** Normally, the model definition provides all possible enumeration options for an attribute of the type enumeration. Typical enumeration options contain less than ten options to select from. In the medical context, clinicians want to prescribe drugs to patients. The effort to model and maintain all drugs as enumeration options is not justifiable, considering that databases with all currently available drugs exist. An alternative option would be to synchronize all drugs as entities and to link those, but the maintainability issue remains. Representing drugs as a string attribute value is not sufficient, because it does not allow automatic processing after the prescription is performed. Therefore, the user interface supports representing a json attribute value as an external enumeration value. If an enumeration option is selected, the attribute json value contains a name and the corresponding id. The user interface supports representing a json attribute value containing an id and a corresponding name as enumeration option that is searchable. The external enumeration model element provides a URL with search placeholders to execute the actual user interface filter. The external enumeration pattern can be applied to several other scenarios.

- **Public Link** A string that is interpreted and represented as a hyperlink pointing to any publicly available HTTP resource.
- **Private Link** Assuming that multiple micro-services use a common SSO and want to reference private resources, the request needs to contain an authentication token. Besides the authentication token, the private link works similar to the public link and can point to any HTTP/HTTPS resource that uses the same authentication token as the frontend.
- **Hidden Flag** The case data page by default shows all entity attributes. Complex calculations may be decomposed into several derived attributes, but only one represents the final result relevant for the user interface representation. A complex calculation may be the dynamic calculation of the body representation, as shown in Figure 5.9, where one attribute contains the plain template and another derived attribute uses the template attribute and injects all the needed variables. The hidden flag is only considered in the data page, the task representation and the summary representation ignore this flag because the attribute could be removed from the task or summary model.

The custom data representations presented above can be annotated on the attribute definition model. In general, all representations related to an attribute can apply the customizations. This especially includes the task representation, the data page, and the summary page.



Patient Questionaires *

State: COMPLETED

Last Modification: Jan 16, 2019 11:09

MOBILITY *
I have slight problems in walking about

SELF-CARE * ?
I have moderate problems washing or dressing myself

USUAL ACTIVITIES *
I am unable to do my usual activities

PAIN / DISCOMFORT *
I have moderate pain or discomfort

ANXIETY / DEPRESSION *
I am severely anxious or depressed

YOUR HEALTH TODAY *
72

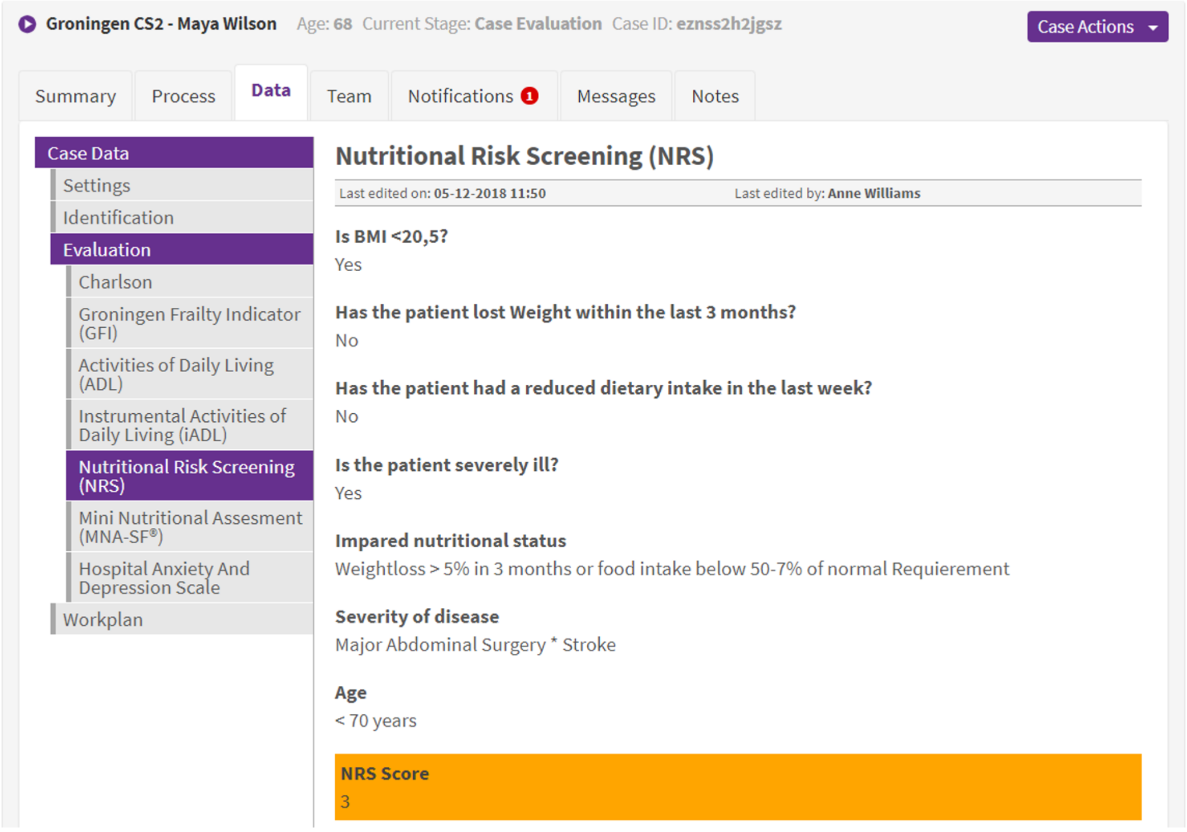
Figure 5.9.: `CustomDataRepresentation` showing a JSON attribute interpreted as a patient questionnaire on the left and a derived string attribute interpreted as an SVG image.

5.1.10. Case Data

The case data shows the resulting linked data structure from the case execution. The execution of every case task containing a writable field creates or modifies the related case data. Figure 5.10 shows the case data page. On the left, a navigation bar is presented, and on the right, a detail view of the currently selected entity is presented.

The first navigation element on the left represents the case root entity that links to their child entities via attributes. These entities recursively link to their child elements, and so forth. When a process element is repeated, a new entity is mostly created for each iteration which leads to attributes linking to multiple entities. E.g., if the case evaluation was repeated, an artificial nesting level allows navigating through the first to the n-th iteration of the case evaluation. The introduction of an artificial nesting level for occurring iterations helps to simplify the navigation of more extensive cases. The linkage path to the currently visible entity is highlighted in purple.

On the right, the currently selected entity is shown. At the top, the entity title is presented, followed by the entity meta-data and entity attributes. The meta-data block contains the last edit date and the editor's first and last name. An edit operation is the instantiation of the entity itself or of the modification of entity attributes. A newly instantiated entity is linked by



Groningen CS2 - Maya Wilson Age: 68 Current Stage: Case Evaluation Case ID: eznss2h2jgsz

Case Actions
Summary
Process
Data
Team
Notifications 1
Messages
Notes
Case Data
Settings
Identification
Evaluation
Charlson
Groningen Frailty Indicator (GFI)
Activities of Daily Living (ADL)
Instrumental Activities of Daily Living (iADL)
Nutritional Risk Screening (NRS)
Mini Nutritional Assesment (MNA-SF®)
Hospital Anxiety And Depression Scale
Workplan

Nutritional Risk Screening (NRS)

Last edited on: 05-12-2018 11:50 Last edited by: Anne Williams

Is BMI <20,5?
Yes

Has the patient lost Weight within the last 3 months?
No

Has the patient had a reduced dietary intake in the last week?
No

Is the patient severely ill?
Yes

Impaired nutritional status
Weightloss > 5% in 3 months or food intake below 50-7% of normal Requierement

Severity of disease
Major Abdominal Surgery * Stroke

Age
< 70 years

NRS Score
3

Figure 5.10.: Case data page showing the linked entity structure of a case.

adding a new attribute value on the parent entity. The main content of the page shows the entity attributes and their related attribute values. While the number and type of attributes depend on the entity definition, the representation of a single attribute is mostly similar to the task representation (cf. Section 5.1.8). Custom data representation as described in Section 5.1.9 is applied to the task representation as well as to the data page. Layout positions are not supported on the data page. Entity attributes might be linked to multiple tasks which could lead to competing layouts.

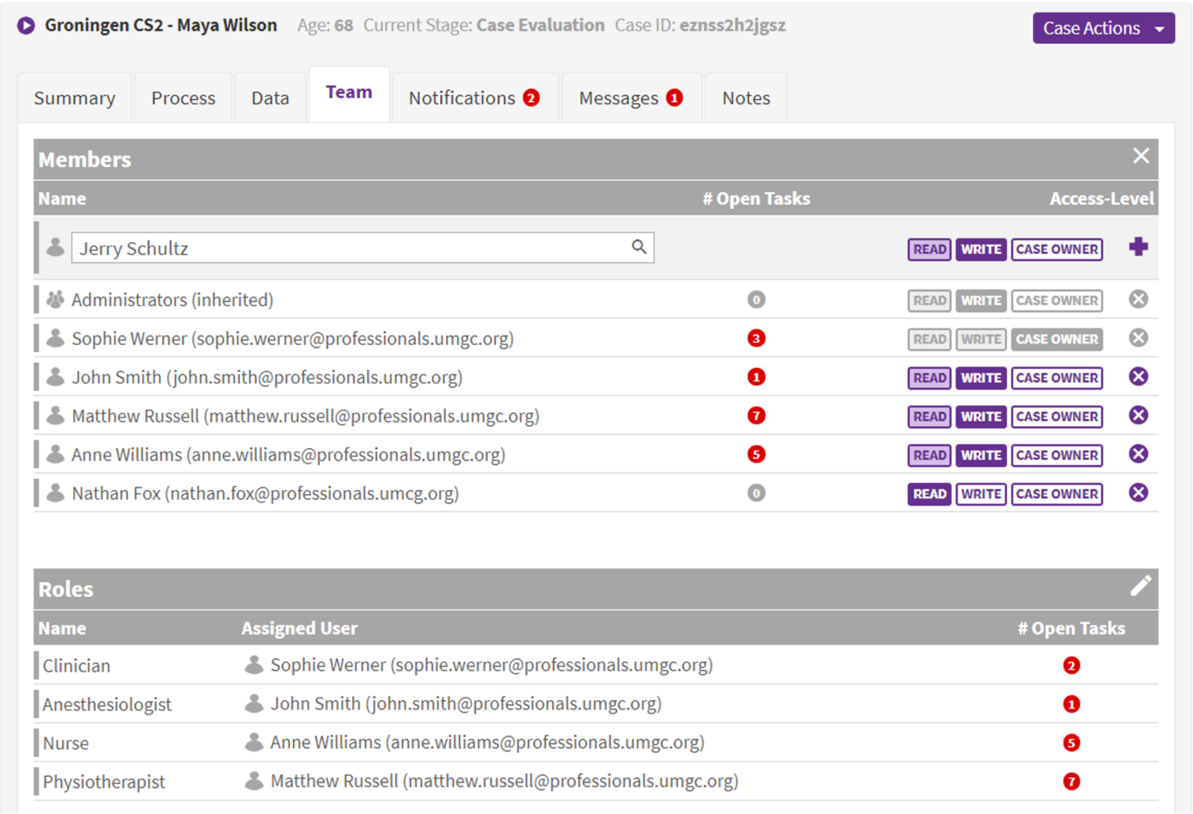
5.1.11. Case Team

A best practice modeling pattern defines a `HumanTask` that needs to be accomplished directly after the case instantiation to initially set the case roles. Typically, sentries are used to prevent the continuation of the case before this task is completed. Enforcing that a case-worker assigns the roles in an early stage increases the collaboration significantly, considering that each activated task for an assigned role is directly visible on the task owner dashboard. However, during the case-execution, roles may need to be adjusted. Additional `HumanTasks` might be used to handle predictable role changes, such as changing or adding a role after a certain stage. The case team page enables managing the case roles and case member access rights dynamically at any time, as illustrated in Figure 5.11.

The first block lists the case members, each with the specific access level that has been assigned to them. A member is either a group or a user, which is indicated with an icon. For a user, the first and last name is presented. The email in brackets behind allow identifying users uniquely. For a group, the related group name is presented. Inherited user or group rights from the workspace are indicated with a label named inherited within brackets. Classically, an administrator group is inherited from the workspace to allow administrators accessing all existing cases for a workspace. Similarly, read-only access for all cases within a workspace could be granted to a secretary. Possible explicit access levels are: i) read-only access, e.g., for administrative users, ii) write access which allows performing most system actions, such as completing a task, correcting a task, writing a message, editing the notes, editing the members or roles, iii) case owner access which allows to terminate a case, complete a case or even delete a case. In addition to the explicit access level, each case member implicitly has all access rights of the lower access levels. Explicit access rights are represented with a purple background and implicit access rights with a lighter background. A case can have multiple readers and writers but only one case owner. All case members receive messages that are visible on their dashboard until a message is marked as read. Therefore, the action marking a message as read is also allowed with the read-only access level. The open- tasks column shows a counter of the currently active tasks which are assigned to that user. This information is essential when removing a case member, considering that this task is unassigned but still pending, or when downgrading a user to read-only access because that user will then not able to accomplish this task with read-only access.

For reason of usability, the team page is shown in read-only mode by default. Every case member with write access can switch into edit mode by clicking the edit pencil at the top right corner. The team members block shows an additional row with a gray background to add new users. A search allows finding current workspace users that might be added. The default access level is set to read-only and can be changed by clicking on another access level. Finally, the plus button allows adding the user with the defined access level as a case member. The access rights of existing members can be adapted by clicking on another access level or completely revoked by clicking on the cross button. Inherited access rights are not modifiable and are represented as faded out during the editing. The case owner access right cannot be revoked. Instead, the case owner access is be assigned to another member and is then automatically revoked from the previous case owner. This mechanism ensures there is exactly one case owner who is responsible and able to manage the case at any time. Due to the side effects caused by implicit changes, all member changes are treated as atomic actions to prevent issues.

The second block lists all case roles with the currently assigned users. The visible roles are dynamically extracted from the case template. All process definitions that declare an owner link to an attribute which implicitly declares a role. Each role is represented with the attribute name, the assigned user including their email address, and a number of currently pending tasks for this role. The edit button at the top right corner enables switching into edit mode. Each role might be changed with a searchable input field comparable with the add member functionality. An attribute that represents a role mostly has a group constraint, which means only direct or indirect members of this group can be selected for the role. All users who are assigned to a role receive write access granted to ensure pending tasks can be accomplished.



Groningen CS2 - Maya Wilson Age: 68 Current Stage: Case Evaluation Case ID: eznss2h2jgsz

Members

Name	# Open Tasks	Access-Level
Jerry Schultz		READ WRITE CASE OWNER
Administrators (inherited)	0	READ WRITE CASE OWNER
Sophie Werner (sophie.werner@professionals.umgc.org)	3	READ WRITE CASE OWNER
John Smith (john.smith@professionals.umgc.org)	1	READ WRITE CASE OWNER
Matthew Russell (matthew.russell@professionals.umgc.org)	7	READ WRITE CASE OWNER
Anne Williams (anne.williams@professionals.umgc.org)	5	READ WRITE CASE OWNER
Nathan Fox (nathan.fox@professionals.umcg.org)	0	READ WRITE CASE OWNER

Roles

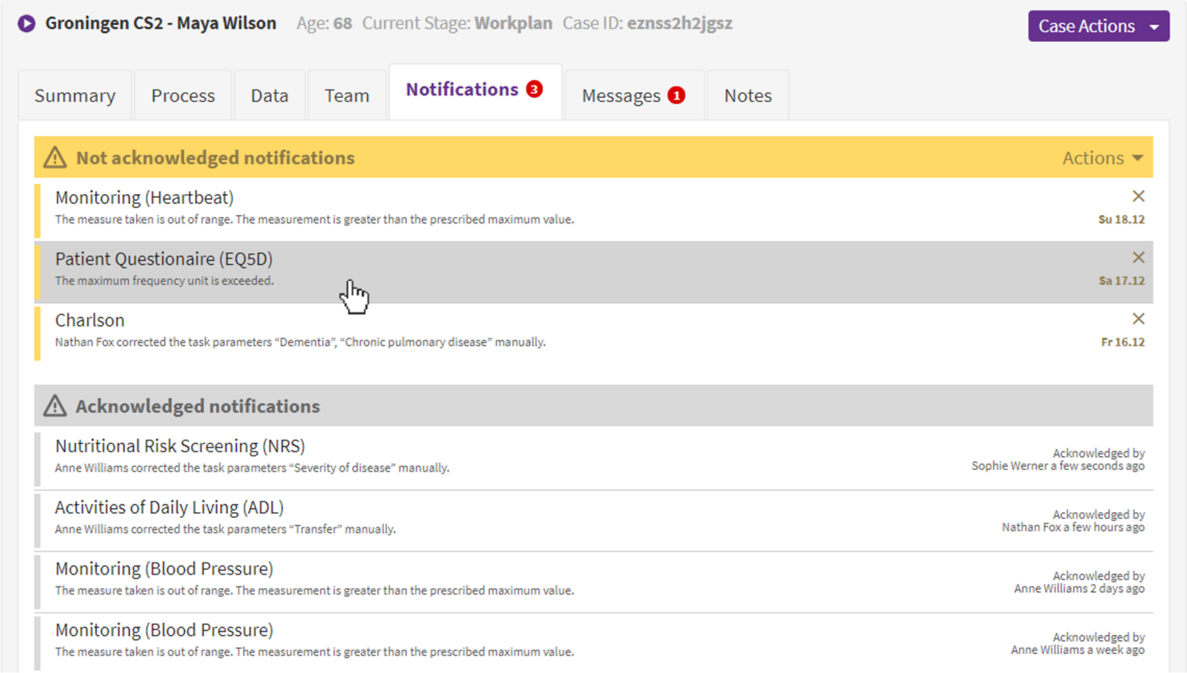
Name	Assigned User	# Open Tasks
Clinician	Sophie Werner (sophie.werner@professionals.umgc.org)	2
Anesthesiologist	John Smith (john.smith@professionals.umgc.org)	1
Nurse	Anne Williams (anne.williams@professionals.umgc.org)	5
Physiotherapist	Matthew Russell (matthew.russell@professionals.umgc.org)	7

Figure 5.11.: Case team showing the case members and case roles.

5.1.12. Case Notifications

The case notification tab shows all notifications related to a case, as illustrated in Figure 5.12. On top, all non-acknowledged notifications are represented as a preview colored in yellow and on the bottom, all acknowledged notifications are shown as a preview colored in gray. Each notification is represented with the process title and shows a notification text preview in the line below. Clicking on a notification opens the task detail page which shows all notification details in the related process context where the notification occurred. Non-acknowledged notifications show the date of occurrence on the right and allow to quickly acknowledge the notification by clicking on the cross. A contextual menu conveniently allows to acknowledge all notifications with a single interaction. All acknowledged notifications show the first and last name of the users who acknowledged the notification and the related date to enable tracking the changes. To indicate occurring notifications within the case before the case notification tab is opened, a red counter shows the number of non-acknowledged notifications at the top.

The non-acknowledged notifications are visible on the user's individual dashboards, indicated on the my-case view, shown on the case workflow, and shown in detail on the task representation page to indicate needed contributions. Notifications are only shown on the dashboard of the related task owner and the related case owner to focus on responsible users who primarily need to contribute and to prevent an unnecessary information flood. To enable working on a case collaboratively, all other views are simply showing the notifications independent of the task and case ownership. The collaboration aspect allows that all non-acknowledged notifications can be acknowledged by every user who has case access rights. To ensure traceability, the user who acknowledges a notification is documented.



⚠ Not acknowledged notifications

Monitoring (Heartbeat)
The measure taken is out of range. The measurement is greater than the prescribed maximum value.
Su 18.12

Patient Questionaire (EQ5D)
The maximum frequency unit is exceeded.
Sa 17.12

Charlson
Nathan Fox corrected the task parameters "Dementia", "Chronic pulmonary disease" manually.
Fr 16.12

⚠ Acknowledged notifications

Nutritional Risk Screening (NRS)
Anne Williams corrected the task parameters "Severity of disease" manually.
Acknowledged by Sophie Werner a few seconds ago

Activities of Daily Living (ADL)
Anne Williams corrected the task parameters "Transfer" manually.
Acknowledged by Nathan Fox a few hours ago

Monitoring (Blood Pressure)
The measure taken is out of range. The measurement is greater than the prescribed maximum value.
Acknowledged by Anne Williams 2 days ago

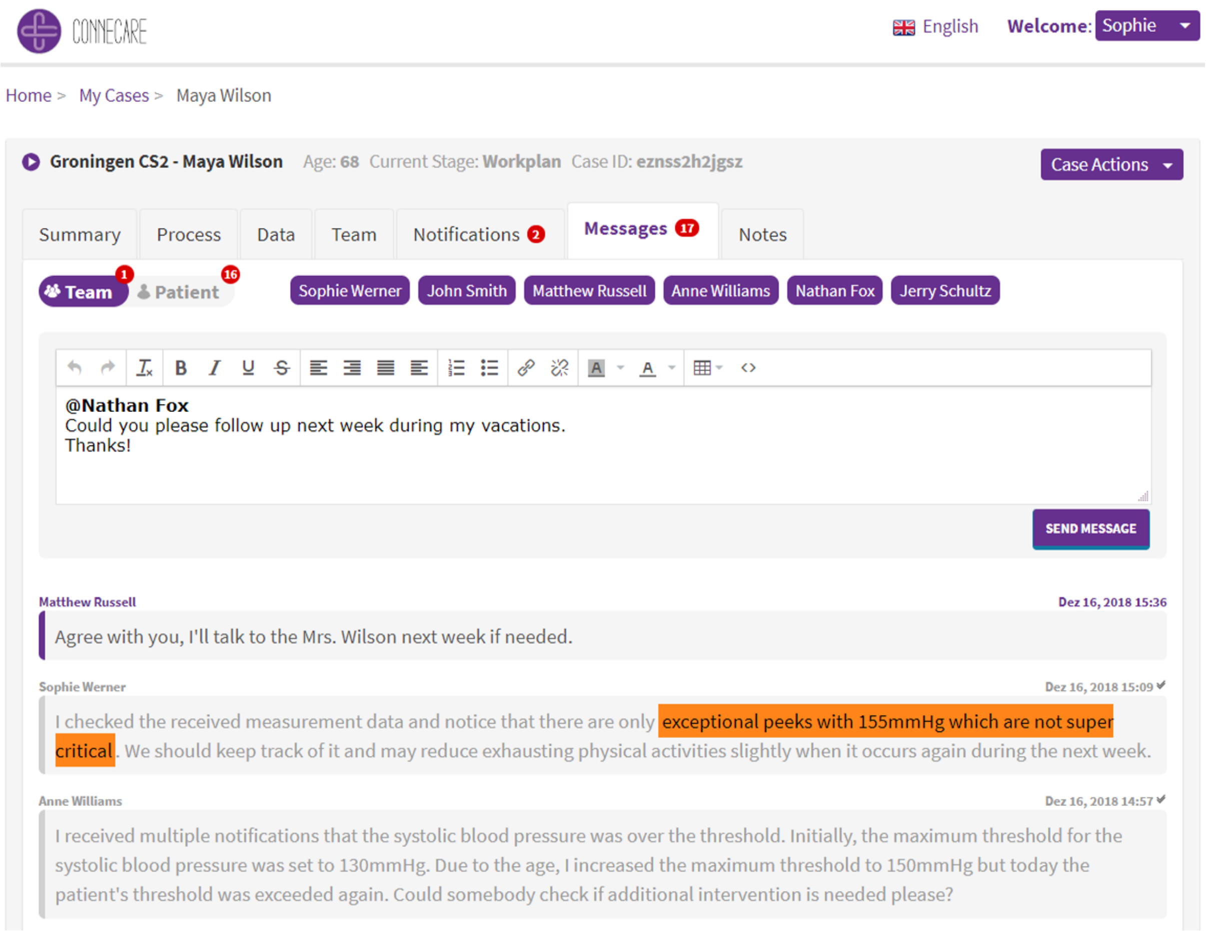
Monitoring (Blood Pressure)
The measure taken is out of range. The measurement is greater than the prescribed maximum value.
Acknowledged by Anne Williams a week ago

Figure 5.12.: Case notifications show all notifications of the case.

5.1.13. Case Messages

The treatment of a patient includes knowledge-intensive tasks that often require communication with other care professionals for knowledge exchange. Therefore, the user interface supports case-based messaging from professionals to professionals, as illustrated in Figure 5.13 and integrates the professional to patient communication with third-party endpoints. A red counter indicates the number of unread messages on the messaging tab that contains three conceptual blocks.

The first block indicates with a switch whether the team messaging or the patient messaging is active. Red counters indicate the number of unread messages. On the left, a list of all potential recipients for a new message is shown. The list contains all members represented in the case team tab. The second block allows sending a new message in the currently visible conversation. A What You See Is What You Get (WYSIWYG) editor allows to prepare a message with rich text elements, such as bold text, highlighted text, or several other options as indicated with the icons of the editor. The third block shows all messages of the current conversation in reverse chronological order. For every message, meta-data is illustrated, such as the author, the send



Groningen CS2 - Maya Wilson Age: 68 Current Stage: Workplan Case ID: eznss2h2jgsz

@Nathan Fox
Could you please follow up next week during my vacations.
Thanks!

Matthew Russell — Dez 16, 2018 15:36

Agree with you, I'll talk to the Mrs. Wilson next week if needed.

Sophie Werner — Dez 16, 2018 15:09 ✓

I checked the received measurement data and notice that there are only exceptional peeks with 155mmHg which are not super critical. We should keep track of it and may reduce exhausting physical activities slightly when it occurs again during the next week.

Anne Williams — Dez 16, 2018 14:57 ✓

I received multiple notifications that the systolic blood pressure was over the threshold. Initially, the maximum threshold for the systolic blood pressure was set to 130mmHg. Due to the age, I increased the maximum threshold to 150mmHg but today the patient's threshold was exceeded again. Could somebody check if additional intervention is needed please?

Figure 5.13.: Case messages showing a conversion between care professionals.

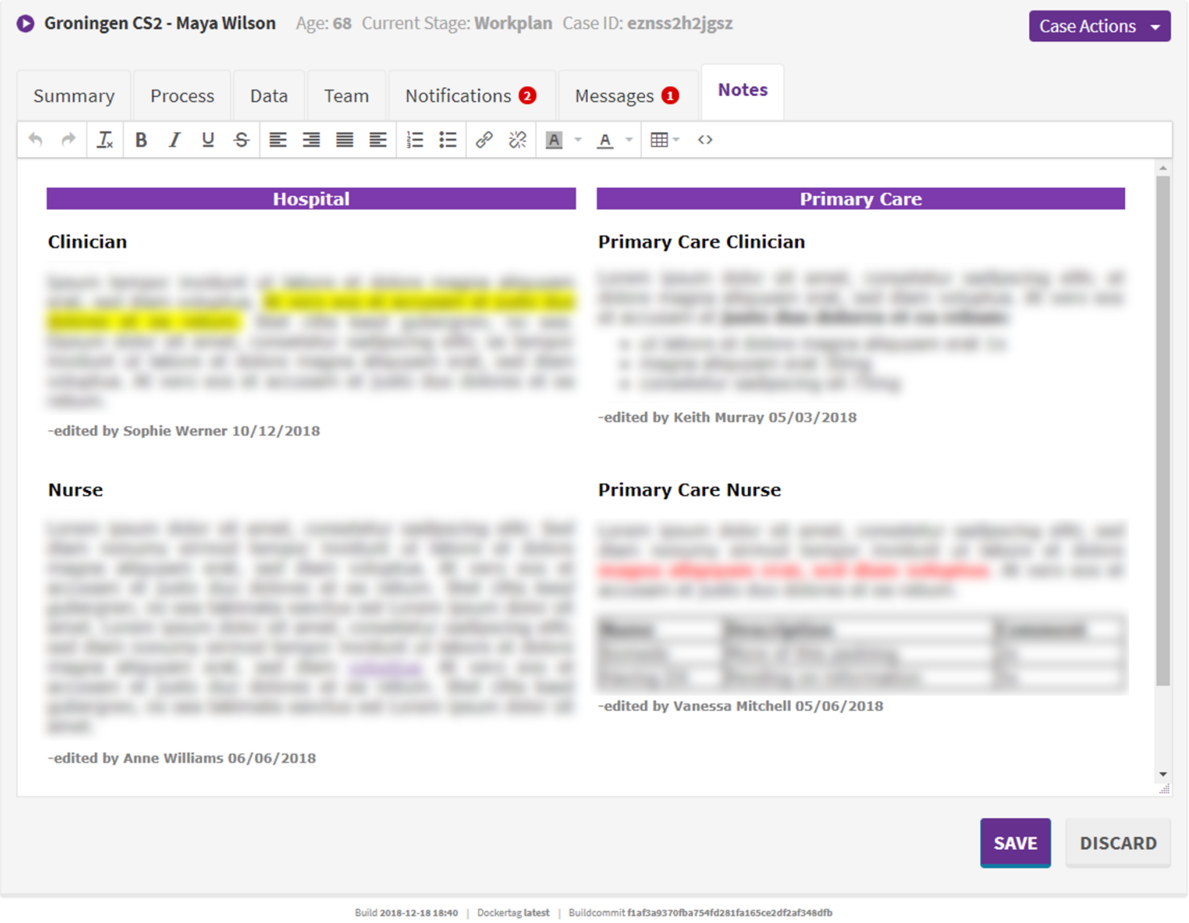
date, and the is-read flag. Unread messages are indicated with a purple border on the left and a purple colored author and sent date. Clicking on a message marks it as read and shows a checkmark behind the send date. If multiple messages are unread, a convenience option allows marking all as read with a single interaction. Each user marks messages individually. This allows removing a message from the dashboard of a user who read the message and to keep a message on the dashboard of a user who has not read the message yet. For users who are newly added as case team members, all old messages appear as unread.

5.1.14. Case Notes

In the context of integrated care, the coordination of ad-hoc information exchange and collaborative documentation between the hospital professionals and the primary care professionals is essential. Figure 5.14 illustrates the case notes page in edit mode. The What You See Is What You Get (WYSIWYG) editor shows the editable rendered HTML content. A table is used to informal structure the information. The first column represents the hospital roles and the second column represents the corresponding primary care roles. Each role is represented with the role name as a headline, with information created by the role, and with a footer indicating when the content was modified by who. The exchanged information can be about medication, complications that occurred in the past, or about any other relevant topic. To structure this information, the most relevant text parts can be marked or colored, links can be used to refer to relevant content, and tables can be used as a structuring element. The usage described above simply illustrates a detailed sample. The unstructured notes page can be used flexibly to cover the need for ad-hoc documentation.

By default, the case notes page is shown in read-only mode and case members who have write access can switch into the edit mode if needed. To prevent concurrent editing, edit tokens are issued when switching into edit mode. When a user saves changes, the backed ensures the edit token contained in the request is equal with the persisted edit token, or otherwise, the update fails. In the read-only mode, a footer indicates when the last edit was performed by which user.

There are definite similarities regarding the usage pattern within a single case definition, which represents a specific treatment plan. Therefore, the case definition model allows the optional definition of an HTML-based template that is initially provided and can be modified during the case execution. Even though the case notes page is designed for unstructured information exchange to complement the structured process execution, templates help to increase usability while ensuring a maximum of flexibility.



Groningen CS2 - Maya Wilson Age: 68 Current Stage: Workplan Case ID: eznss2h2jgsz

Hospital

Clinician

-edited by Sophie Werner 10/12/2018

Nurse

-edited by Anne Williams 06/06/2018

Primary Care

Primary Care Clinician

-edited by Keith Murray 05/03/2018

Primary Care Nurse

-edited by Vanessa Mitchell 05/06/2018

Build 2018-12-18 18:40 | Dockertag latest | Buildcommit f1af3a9370fba754fd281fa165ce2df2af348dfb

Figure 5.14.: Case notes enable ah-hoc documentation of unstructured content.

5.1.15. User and Role Management

The user and role management is accessible from the dashboard and allows managing all system users. In the integrated care context, there are two types of users: care professionals and patients. The user interface shows four tabs: create patient, edit patient, create care professional, and edit care professional.

Creating a patient-user requires information such as an email, a referencing patient number, the first and last name, the date of birth, the marital status, the education level, the socio-cultural level, the phone number, and the preferred language. The password is set by default and can be changed by patients afterward. The user identity management system automatically assigns a role to each newly created patient which identifies them as a patient. Editing a patient first requires searching the patient, selecting the desired patient from the resulting patient list, and then the patient can be edited.

Creating a care professional user requires different information, including their email, first and last name, the preferred language, and the desired set of roles. By default, globally available roles of the user identity management system are admin, admin-officer, anesthesiologist, case-manager, clinician, data-manager, lab-technician, local-pulmonologist, nurse, nutritionist, physician, physiotherapist, primary care nurse, primary care clinician, psychologist, secretary, and social carer. The process of editing a care professional user is similar to editing a patient. The password of the currently logged-in care professional user can be changed in the user's context menu placed at the top of the page.

Considering that the system is deployed in an integrated care context and that the non-model-based user identity management acts as a single point of truth regarding the user and role management, this part of the user interface is hardcoded as well.

5.2. Case Model Definition Reference

This section describes the XML case template structure that is used for modeling case definitions. This XML representation is transformed in sequential REST API requests which create a case definition within the ACM4IC engine (cf. Section 5.4). For reason of usability, the XML representation slightly differs compared to the API resources.

Figure 5.15 illustrates the XML structure in a UML-like notation that is used to configure a newly deployed ACM4IC engine. Each graphical node represents an XML element colored according to the conceptual layer (cf. Section 4.1). In the visual notation, nested XML elements are attached to their parent XML element. The XML root element is named `ACM4IC` and contains all modeling elements. According to our best practice, a workspace file starts with the `Settings` element declaring global rights where technically, the order does not matter. The `UserDefinition` allows modeling the `AttributeDefinitions` and `DerivedAttributeDefinitions` to customize the `User` object. With the `User` and related `Attribute` element, corresponding user accounts are created with the declared values. Desired `Groups`, including their `Administrators` and

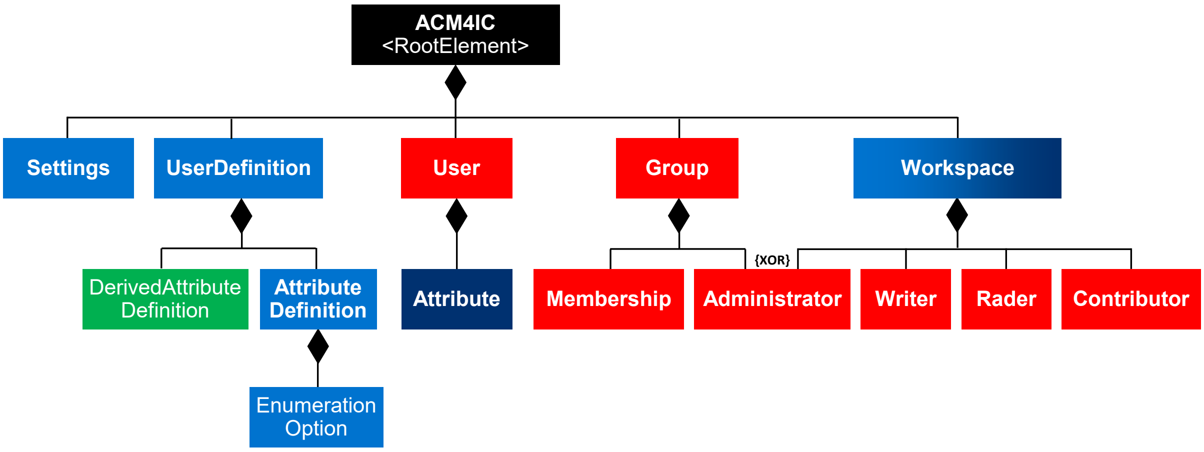


Figure 5.15.: XML elements representing the workspace file with a UML-like notation.

Memberships can be modeled. Declaring a Membership allows assigning a previously declared User or Group as a member of a specific Group. Finally, the Workspace element is declared, which most likely uses all previously declared elements to assign the Administrators, Writers, Readers, and Contributors. Declared Workspace Readers and Writers automatically receive inherited access to all instantiated cases.

Figure 5.16 illustrates the XML structure defining a case template. Similar to the workspace file, the case template file starts with the XML root element ACM4IC. A referencing pattern allows reusing global model elements. Common referenced modeling elements such as Groups, Users, and Workspaces are declared within the workspace file and referenceable within the case template file. Elements that declare a reference to already imported modeling elements are indicated with a star. Static primary identifiers matching the database identifiers must be declared to allow referencing existing objects. The Workspace contains all modeling elements needed to declare a case template. First, the required schemata are defined with the EntityDefinitions, then a CaseDefinition is modeled using the declared schemata. EntityDefinitions allow declaring data schemata using the AttributeDefinitions and if desired DerivedAttributeDefinitions. With the exception of the EnumerationOptions, all constraints are declarable within the AttributeDefinition element. Within the CaseDefinition, all process-centric meta-model elements are declared. The SummarySectionDefinition in combination with multiple containing SummaryParamDefinitions allows declaring template-specific summaries depending on the data layer. The crucial CMMN execution semantics is declared with specialized ProcessDefinitions and TaskDefinitions. Those abstract elements are in-

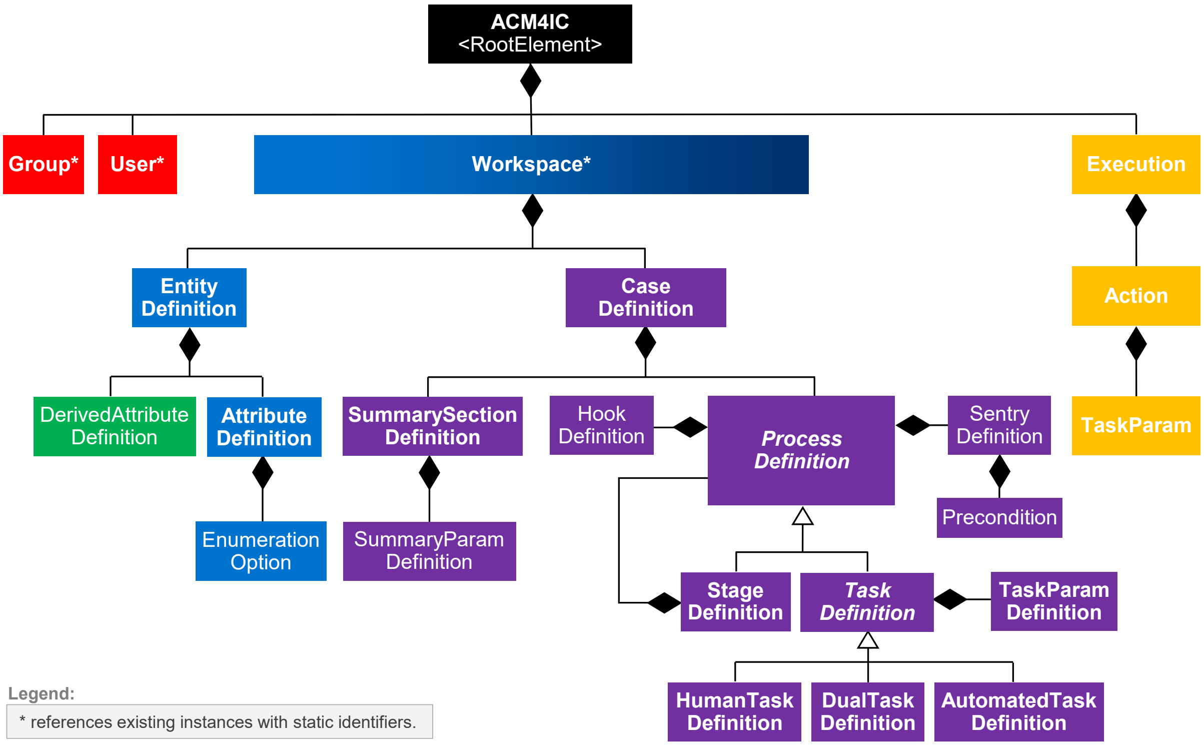


Figure 5.16.: XML elements representing the case template file with a UML-like notation.

dicated with italic names and do not exist as XML modeling elements. They are merely required to specify the conceptual XML file structure. A `StageDefinition` can contain multiple `HumanTaskDefinitions`, `DualTaskDefinitions`, or `AutomatedTaskDefinitions`. Each `TaskDefinition` can specify the input and output with multiple `TaskParamDefinitions` linking to the data layer. All specialized `ProcessDefinitions` support declaring `SentryDefinitions` with multiple `Preconditions` referencing to `ProcessDefinitions`. Similarly, `HookDefinitions` are declarable. Declarations can quickly become complex. Therefore, an execution flow can be declared for an automated test `Execution`. Within the `Execution` scope, typical execution `Actions` are declarable, such as activate, complete, and correct a task. Optionally, the declaration of execution breakpoints is supported to debug complex models stepwise.

■ Referencing Elements (Group*, User*, Workspace*)

Referencing elements are used to link to existing model elements from the database. The initial `Users`, `Groups`, and `Workspaces` are created globally, typically directly after the deployment is completed. Multiple `CaseDefinitions` are assigned to one `Workspace`. Therefore, the existing `Workspace` must be referenced within the XML case template file. `Groups` can be referenced as owner constraints for the `CaseDefinition` or as owner constraints of a `TaskDefinition`. A `User` might be referenced during the test execution as `TaskParam` value.

Attribute	Required	Description
`id`	mandatory	Declares a human-readable `id` assigned by the modeler of the referencing model element. The id is used to resolve XML references during the import.
`staticId`	mandatory	Declares a `staticId` representing the identifier from the database.

Default values are not supported!

XML Example:

```
<!-- The user with the identifier "WilsonG" within the XML file references the existing user with the
     identifier "f854153e05b111e8941d0242ac140002" from the case execution engine database -->
<User staticId="f854153e05b111e8941d0242ac140002" id="WilsonG"/>

<!-- The group with the identifier "MunichProfessionals" within the XML file references the existing group
      with the identifier "2c9480885d1737ef015d74deecbf0004" from the case execution engine database -->
<Group staticId="2c9480885d1737ef015d74deecbf0004" id="MunichProfessionals"/>

<!-- The workspace with the identifier "Munich" within the XML file references the existing workspace with
      the identifier "2c9480885d1737ef015d74deecbf0004" from the case execution engine database -->
<Workspace staticId="2c9480885d1737ef015d74deecbf0004" id="Munich">
```

User

The `UserDefinition` allows declaring a schema with `AttributeDefinitions` and `DerivedAttributeDefinitions` similar to the `EntityDefinition` and is therefore not further specified. The generic `User` information, such as name and email are declared as XML attributes and the `UserDefinition`-dependent information with XML sub-elements named `Attributes`. The email must be unique within the deployed instance, whereas the name does not have to be unique. For reason of privacy, the user's password cannot be declared within the XML file.

Attribute	Required	Description
`id`	mandatory	Declares the `id` used to resolve references within the XML declaration and is replaced with a dynamically generated id.
`staticId`	optional	Declares a `staticId` that is used as a unique identifier within the database. It allows using externally generated identifiers with up to 32 characters but the modeler must ensure that the declared identifier is still available. Otherwise, the import fails.
`name`	mandatory	Declares the `name` that is visible in the user interface. Typically, the `name` is used to label auto-complete options.
`email`	mandatory	Declares the `email`, but the uniqueness across all `Users` must be ensured otherwise the import fails.

Default values are not supported!

XML Example:

```xml
<UserDefinition>
  <AttributeDefinition id="firstname" type="string" multiplicity="maximalOne" description="Firstname"/>
  <AttributeDefinition id="lastname" type="string" multiplicity="maximalOne" description="Lastname"/>
  <AttributeDefinition id="birthdate" type="date" multiplicity="maximalOne" description="Birthdate"/>
  <DerivedAttributeDefinition id="age" expression="floor((Today - birthdate)/365)" description="Age"/>
  <AttributeDefinition id="gender" type="enumeration" multiplicity="maximalOne" description="Gender">
    <EnumerationOption value="MALE" description="Male"/>
    <EnumerationOption value="FEMALE" description="Female"/>
  </AttributeDefinition>
</UserDefinition>

<User id="SchultzJ" staticId="2c940c085e7650" name="Jerry Schultz" email="jerry.schultz@gmail.com">
  <Attribute attributeDefinitionId="firstname" values="['Jerry']"/>
  <Attribute attributeDefinitionId="lastname" values="['Schultz']"/>
  <Attribute attributeDefinitionId="birthdate" values="['1980-03-08T00:00:00.000']"/>
  <!-- The age attribute is computed automatically based on the birth date -->
  <Attribute attributeDefinitionId="gender" values="['MALE']"/>
</User>

<!-- add more user declarations here -->
```

Group

The `Group` declares flat information such as the name as an XML attribute and supports declaring multiple `Administrators` and `Memberships` as XML sub-elements. The `Membership` references a principal which might be a `Group`, and this allows declaring nested `Groups`. Each `Group` must declare at least one `Administrator` element.

Attribute	Required	Description
`id`	mandatory	Declares the `id` used to resolve references within the XML declaration and is replaced with a dynamically generated id.
`staticId`	optional	Declares a `staticId` that is used as a unique identifier within the database. It allows using externally generated identifiers with up to 32 characters but the modeler must ensure that the declared identifier is still available. Otherwise, the import fails.
`name`	mandatory	Declares the `name` that is visible in the user interface. Typically, the `name` is used to label auto-complete options.
`description`	mandatory	Declares the `description` shown on the user interface and is typically expressed in the local language.

Default values are not supported!

XML Example:

```xml
<Group id="MunichPatients" staticId="2c9480845bee03e7015bfc056b070002"  name="Patients"
    description="Patients">
  <Administrator principalId="MunichProfessionals"/> <!-- All professionals are administrators -->
  <Membership principalId="HopkinsC"/>
  <Membership principalId="HayesK"/>
  <Membership principalId="SchultzJ"/>
</Group>

<Group id="MunichProfessionals" staticId="2c9480845bee03e7015bfc03da610002" name="Professionals"
    description="Professionals">
  <Administrator principalId="Me"/> <!-- Me references the user importing the declaration -->
  <Membership principalId="MunichCaseManagers"/>
  <Membership principalId="MunichNurses"/> <!-- Nest the group nurses below the professionals -->
  <Membership principalId="MunichPhysicians"/>
  <Membership principalId="Munichphysiotherapists"/>
</Group>
```

Membership, Administrator, Writer, Reader and Contributor

The `Group Membership`, `Group Administrator`, `Workspace Administrator`, `Workspace Writer`, `Workspace Reader`, and `Workspace Contributor` are declared analogically by referencing a `principalId`. Multiple principals are assigned with multiple XML elements. Holistic samples are presented near the `Group` and `Workspace` modeling.

Attribute	Required	Description
`principalId`	mandatory	Declares the `principalId` referencing to a `Principal` which should be assigned.

Default values are not supported!

Workspace

The generic Workspace information is such as the description is declared as an XML attribute and the access rights are specified with XML sub-elements. A Workspace must declare at least one Administrator and can optionally configure Writer, Reader, and Contributor which refer to a principal. Nested Groups are considered implicitly.

Attribute	Required	Description
id	mandatory	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
staticId	optional	Declares a staticId that is used as a unique identifier within the database. It allows using externally generated identifiers with up to 32 characters but the modeler must ensure that the declared identifier is still available. Otherwise, the import fails.
description	mandatory	Declares the description shown on the user interface and is typically expressed in the local language.

Default values are not supported!

XML Example:

```xml
<Workspace id="Munich" staticId="2c9480845bee03e7015bfc03da610002" description="Klinikum rechts der Isar">

  <!-- Declares the global predefined administrator's group as administrator -->
  <Administrator principalId="Administrators"/>

  <!-- Grants inherited write access for all cases within the workspace to all Munich administrators -->
  <Writer principalId="MunichAdministrators"/>

  <!-- Grants inherited read access for all cases within the workspace to all Munich secretaries -->
  <Reader principalId="MunichSecretaries"/>

  <!-- Grants principals the right to instantiate new cases without having read or write access for the
     overall workspace. A user who instantiates a case automatically receives write access for that case
     and additional case readers and writers may be added later. The contributor concept enables powerful
     dedicated access rights which are typically required for knowledge-intensive processes. -->
  <Contributor principalId="MunichProfessionals"/>

  <!-- place <EntityDefinition> ... here -->
  <!-- place <CaseDefinition> ... here -->

</Workspace>
```

EntityDefinition

The EntityDefinition, in combination with the AttributeDefinition and DerivedAttributeDefinition, enables declaring a schema similar to a UML class. The example below illustrates declaring a schema for a medical questionnaire named Charlson Comorbidity Index, as illustrated in Figure 5.7. The first EntityDefinition illustrates the parent schema that links to the Charlson EntityDefinition. The EntityDefinition contains multiple AttributeDefinitions and DerivedAttributeDefinitions to model expected input data and to dynamically calculate a score. Additionally, the EntityDefinition contains an SVG template declaration, dynamically calculated SVG styles, and the resulting automatically generated SVG representation.

Attribute	Required	Description
id	mandatory	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
description	mandatory	Declares the description shown on the user interface and is typically expressed in the local language.

Default values are not supported!

XML Example:

```xml
<EntityDefinition id="MCS2_Identification" description="Identification">
  <AttributeDefinition id="MCS2_Charlson" description="Charlson Comorbidity Index" type="Link.
     EntityDefinition.MCS2_Charlson" multiplicity="any" />
  <AttributeDefinition id="MCS2_Barthel" description="Barthel" type="Link.EntityDefinition.MCS2_Barthel"
     multiplicity="any" />
  <!-- Place additional <AttributeDefinition> to extend evaluation schema here -->
</EntityDefinition>

<EntityDefinition id="MCS2_Charlson" description="Charlson Comorbidity Index">
  <AttributeDefinition id="inclusiondiag" type="enumeration" multiplicity="exactlyOne" description="1.
     Diagnostics of Surgery">
    <EnumerationOption value="LKP" description="Left knee prosthesis (LKP)"/>
    <EnumerationOption value="RKP" description="Right knee prosthesis (RKP)"/>
    <EnumerationOption value="LHA" description="Left hip arthroplasty (LHA)"/>
    <EnumerationOption value="RHA" description="Right hip arthroplasty (RHA)"/>
  </AttributeDefinition>
  <AttributeDefinition id="ch1" type="enumeration" multiplicity="exactlyOne" description="2. Myocardial
     infarction" additionalDescription="Most common symptom is chest pain.">
    <EnumerationOption value="0" description="No"/>
    <EnumerationOption value="1" description="Yes"/>
  </AttributeDefinition>
  <AttributeDefinition id="ch2" type="enumeration" multiplicity="exactlyOne" description="3. Congestive
     heart failure">
    <EnumerationOption value="0" description="No"/>
    <EnumerationOption value="1" description="Yes"/>
  </AttributeDefinition>
  <!-- Place additional <AttributeDefinition> here -->
  <DerivedAttributeDefinition id="ch21" description="Charlson" expression="number(ch1,0) + number(ch2,0) +
      ... + number(ch11,0)*2 + ... + number(ch17,0)*3 + number(ch18,0)*6 + ..." uiReference="colors(1<=
     green<=2<orange<=4<red<100)"/>
  <AttributeDefinition id="bodytemplate" description="Body Template" type="string" multiplicity="
     exactlyOne" uiReference="hidden" defaultValue="... declare svg template here ..." />
  <DerivedAttributeDefinition id="bodystyle" explicitType="String" expression=’let green = "#70ad47" in
     ... let red = "#ff0000" in  if ch1="1" or ch2="1" then "#heartfillcolor{fill:"+red+";} " else "#
     heartfillcolor{fill:"+green+";} " ...’ description="Charlson" uiReference="hidden"/>
  <DerivedAttributeDefinition id="bodysvg" explicitType="String" expression=’replace(replace(bodytemplate,
      "#dynamicstylevars{}", bodystyle), "#charlsonscorevar", (if ch21=0 then "N.A." else string(ch21)))’
     description="Charlson" uiReference="svg"/>
</EntityDefinition>
```

■ AttributeDefinition

An `AttributeDefinition` is part of an `EntityDefinition` and specifies a schema for an `Attribute`. The schema declaration primarily includes a type, a multiplicity, default values, and an option to override the default representation on the user interface. The process layer links to those `Attributes`, depending on the `AttributeDefinitions` name resulting from the id declaration to enable the data binding. A holistic best practice sample is provided within the `EntityDefinition` specification.

Attribute	Required	Default	Description
`id`	mandatory	-	Declares the `id` used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the `id` is used as a technical human-readable `name` which is typically declared in English to enable debugging.
`description`	mandatory	-	Declares the `description` shown on the user interface and is typically expressed in the local language.
`additionalDescription`	optional	-	Declares the `additionalDescription`, which provides explanatory information on the user interface and is typically expressed in the local language.
`multiplicity`	optional	`any`	Declares the expected `multiplicity` which supports the values `maximalOne`, `exactlyOne`, `atLeastOne`, or `any`.
`uiReference`	optional	-	Declares the `uiReference` which specifies a `CustomDataRepresentation` to allow customizing the default representation on the user interface.
`externalId`	optional	-	Declares an `externalId` to allow mapping with an external system.
`type`	optional	`notype`	Declares the `type` with implicit constraints that supports the values `link`, `notype`, `string`, `longtext`, `boolean`, `number`, `enumeration`, `date`, or `json`. Additionally, explicit constraints are declarable. The `type link` allows declaring references to any `Entity`, an `Entity` of a specific `EntityDefinition`, any `User`, or a `User` who is a member in a set of `Groups`. The `type number` allows declaring a valid range. The `type date` allows specifying explicit temporal dependencies.
`defaultValues`	optional	-	Declares the `defaultValues` which are initially set. For `enumeration type Attributes`, technical `values` must be used.
`defaultValue`	optional	-	Declares exactly one `defaultValue` which is initially set. For `enumeration type Attributes`, a technical `value` must be used. If the XML attribute `defaultValues` is declared, the `defaultValue` is overridden.

🟦 EnumerationOption

An `EnumerationOption` is part of an `AttributeDefinition` and conceptually comparable with the HTML option element declared within the select element. In addition to the option value and description provided in HTML, an supplemental description and an external identifier can optionally be provided.

Attribute	Required	Description
`value`	mandatory	Declares a technical `value` as a string. Typically expressed in English to simplify debugging.
`description`	mandatory	Declares the `description` shown on the user interface and is typically expressed in the local language.
`additionalDescription`	optional	Declares the `additionalDescription`, which provides explanatory information on the user interface and is typically expressed in the local language.
`externalId`	optional	Declares an `externalId` to allow mapping with an external system.

Default values are not supported!

🟩 DerivedAttributeDefinition

A `DerivedAttributeDefinition` is declared as part of an `EntityDefinition` and specifies an expression to derive values. Expressions may contain complex calculations or simply calculate a score from multiple `AttributeDefinitions`. The expressiveness allows the declaration of queries on existing modeling elements. Additional information is presented by Reschenhofer (2017).

Attribute	Required	Description
`id`	mandatory	Declares the `id` used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the `id` is used as a technical human-readable `name` which is typically declared in English to enable debugging.
`description`	mandatory	Declares the `description` shown on the user interface and is typically expressed in the local language.
`additionalDescription`	optional	Declares the `additionalDescription`, which provides explanatory information on the user interface and is typically expressed in the local language.
`expression`	optional	Declares the `expression`, which calculates the derived value dynamically. Typically, scores depending on `Entity Attributes` are calculated.
`explicitAttributeType`	optional	Declares `explicitAttributeType`, which indicates the expected result type to allow recursive expression declarations.
`uiReference`	optional	Declares the `uiReference` which specifies a `CustomDataRepresentation` to allow customizing the default representation on the user interface.
`externalId`	optional	Declares an `externalId` to allow mapping with an external system.

Default values are not supported!

CaseDefinition

A CaseDefinition is declared in the scope of the Workspace and contains multiple modeling elements relevant for Case-related declarations. Adaptive processes are declared with specialized ProcessDefinitions, such as StageDefinitions, HumanTaskDefinitions, DualTaskDefinitions, or AutomatedTaskDefinitions and with their associated declaration elements. A dynamically calculated Case summary is declarable with multiple SummarySectionDefinitions. The CaseDefinition enables declaring meta-data, such as the case owner, the case client, and hooks. Compared to the ProcessDefinition where multiple HttpHookDefinitions for a state change event are declarable, all CaseDefinition hooks are declared as XML in-line attributes and allow declaring maximum one hook for each supported state change event.

Attribute	Required	Description
id	mandatory	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
description	mandatory	Declares the description shown on the user interface and is typically expressed in the local language.
ownerPath	mandatory	Declares the path to the Case owner based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
clientPath	optional	Declares the path to the Case client based on the Case root Entity. In a medical context, the client represents the treated patient. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
rootEntityDefinitionId	mandatory	Declares the EntityDefinition defining the schema for the Case root Entity.
entityDefinitionId	optional	Declares which EntityDefinition is instantiated and attached as Entity to the entityAttachPath.
entityAttachPath	optional	Declares the path where the newly instantiated Entity should be attached based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
notesDefaultValue	optional	Declares a template for the Case notes where the content can be plain text or HTML.
onActivateHttpHookURL	optional	Declares a URL that is requested on activating a Case.
onCompleteHTTPHookURL	optional	Declares a URL that is requested on completing a Case.
onTerminateHTTPHookURL	optional	Declares a URL that is requested on terminating a Case.
onDeleteHTTPHookURL	optional	Declares a URL that is requested on deleting a Case.

Default values are not supported!

XML Example:

```xml
<CaseDefinition
  id="MCS2_Munich"
  description="Munich CS2"
  ownerPath="MCS2_Settings.CaseOwner"
  clientPath="MCS2_Settings.Patient"
  rootEntityDefinitionId="MCS2_CaseData"
  entityDefinitionId="MCS2_Settings"
  entityAttachPath="MCS2_Settings"
  onCompleteHTTPHookURL="http://integration-producer:8081/v1/sacm/case/terminate"
  onTerminateHTTPHookURL="http://integration-producer:8081/v1/sacm/case/terminate"
  onDeleteHTTPHookURL="http://integration-producer:8081/v1/sacm/case/terminate">

  <!-- place <SummarySectionDefinition> ... here -->
  <!-- place <StageDefinition> ... here -->

</CaseDefinition>
```

SummarySectionDefinition

A SummaryParamDefinitions is part of a CaseDefinition and declares a read-only view linking to Attributes or DerivedAttributes on the data layer. Each SummarySectionDefinition has a section description and typically contains multiple SummaryParamDefinitions to reference Attributes or DerivedAttributes on the data layer. To provide a visual attractive Case summary, each SummarySectionDefinition allows declaring different layout options based on a grid (cf. screen in Figure 5.4).

Attribute	Required	Default	Description
id	mandatory	-	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
description	mandatory	-	Declares the description shown on the user interface and is typically expressed in the local language.
position	optional	STRETCHED	Declares the position, used to layout the summary section definitions in a three-column grid. Possible values are STRETCHED that spans a row over all three columns, LEFT, CENTER, or RIGHT.

XML Example:

```xml
<SummarySectionDefinition id="MCS2_Nutritional" description="Nutritional" position="LEFT">
  <SummaryParamDefinition path="MCS2_Evaluation.MCS2_NRS.nrs8"/>
  <SummaryParamDefinition path="MCS2_Evaluation.MCS2_MNASF.mnasf9"/>
</SummarySectionDefinition>
<SummarySectionDefinition id="MCS2_Body" description="Diagnosis" position="CENTER">
  <SummaryParamDefinition path="MCS2_Identification.MCS2_Charlson.bodysvg"/>
</SummarySectionDefinition>
...
```

SummaryParamDefinition

A SummaryParamDefinition is part of the SummarySectionDefinition and references to an Attribute or DerivedAttribute on the data layer. The path resolving considers the last LinkValue for each path section to ensure linking the most up to date data generated by the latest Process iteration. It implies that the linked Attribute instance might change dynamically when the Process starts a new iteration.

Attribute	Required	Description
path	mandatory	Declares a path depending on the Case root Entity that links to an Attribute or DerivedAttribute.

Default values are not supported!

XML Example:

```
<SummaryParamDefinition path="MCS2_Evaluation.MCS2_NRS.nrs8"/>
```

StageDefinition

A StageDefinition declares the Stage behavior. A composite pattern allows nesting StageDefinitions and therefore, a StageDefinition is either part of the CaseDefinition or of their superordinate StageDefinition. Within a StageDefinition all specialized ProcessDefinitions are declarable, this includes a StageDefinition, a HumanTaskDefinition, DualTaskDefinition, or an AutomatedTaskDefinition and their associated declaration elements (cf. screen in Figure 5.5).

Attribute	Required	Default	Description
id	mandatory	-	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
description	mandatory	-	Declares the description shown on the user interface and is typically expressed in the local language.
ownerPath	optional	-	Declares the path to the Stage owner based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
repeatable	optional	ONCE	Declares the repeatability where ONCE allows only one execution, SERIAL allows multiple repetitions in serial order, and PARALLEL allows multiple active repetitions in parallel.

Attribute	Required	Default	Description
`isMandatory`	mandatory	-	Possible values are `TRUE` and `FALSE`.
`activation`	optional	`AUTOMATIC`	Declares the `activation` trigger which is either `MANUAL` by a case-worker, `AUTOMATIC` by the case execution engine, or depending on a declared `EXPRESSION`.
`manualActivationExpression`	optional	-	Declares the expression to be fulfilled for an `EXPRESSION`-based `activation`.
`entityDefinitionId`	optional	-	Declares which `EntityDefinition` is instantiated and attached as an `Entity` to the `entityAttachPath`.
`entityAttachPath`	optional	-	Declares the path where the newly instantiated `Entity` should be attached based on the `Case` root `Entity`. The path needs to be defined based on the `AttributeDefinition` names. Every path section is separated with a dot.
`externalId`	optional	-	Declares an `externalId` to allow mapping with an external system.
`dynamicDescriptionPath`	optional	-	Declares a path to an `Attribute` or `DerivedAttribute`, the value of which is used to extend the `StageDefinition description` dynamically. The path specification depends on the `Case` root `Entity`. Every path section is separated with a dot.

XML Example:

```xml
<StageDefinition
  id="MCS2_CaseIdentification"
  description="Case Identification"
  isMandatory="true"
  repeatable="ONCE"
  entityDefinitionId="MCS2_Identification"
  entityAttachPath="MCS2_Identification">

  <!-- place <HumanTaskDefinition> here--->
  <!-- place <AutomatedTaskDefinition> here--->
  <!-- place <DualTaskDefinition> here--->

</StageDefinition>
```

■ HumanTaskDefinition

A `HumanTaskDefinition` declares a `HumanTask`, which is performed by case-workers and is typically declared as part of a `StageDefinition`. Compared to an `AutomatedTaskDefinition`, a due date is declarable. The data layer binding is declared with `TaskParamDefinitions` that are part of a `HumanTaskDefinition`. The example below references to the nested case data structure that is declared with `EntityDefinitions` and `AttributeDefinitions` respectively, as previously presented (cf. the resulting screen in Figure 5.7).

Attribute	Required	Default	Description
id	mandatory	-	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
description	mandatory	-	Declares the description shown on the user interface and is typically expressed in the local language.
ownerPath	optional	-	Declares the path to the HumanTask owner based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
dueDatePath	optional	-	Declares the path to the HumanTask due date based on the Case root Entity. The path requires to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
repeatable	optional	ONCE	Declares the repeatability where ONCE allows only one execution, SERIAL allows multiple repetitions in serial order, and PARALLEL allows multiple active repetitions in parallel.
isMandatory	mandatory	-	Possible values are TRUE and FALSE.
activation	optional	AUTOMATIC	Declares the activation trigger which is either MANUAL by a case-worker, AUTOMATIC by the case execution engine, or depending on a declared EXPRESSION.
manualActivation Expression	optional	-	Declares the expression to be fulfilled for an EXPRESSION-based activation.
entityDefinitionId	optional	-	Declares which EntityDefinition is instantiated and attached as an Entity to the entityAttachPath.
entityAttachPath	optional	-	Declares the path where the newly instantiated Entity should be attached based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
externalId	optional	-	Declares an externalId to allow mapping with an external system.
dynamicDescription Path	optional	-	Declares a path to an Attribute or DerivedAttribute, the value of which is used to extend the HumanTaskDefinition description dynamically. The path specification depends on the Case root Entity. Every path section is separated with a dot.

XML Example:

```xml
<HumanTaskDefinition
  id="MCS2_Charlson"
  description="Charlson Comorbidity Index"
  ownerPath="MCS2_Settings.Nurse"
  isMandatory="true"
  repeatable="ONCE"
  entityDefinitionId="MCS2_Charlson"
  entityAttachPath="MCS2_Identification.MCS2_Charlson"
  footnote="Mary E. Charlson, Peter Pompei, Kathy L. Ales & C.Ronald MacKenzie (1987). A new method of
     classifying prognostic comorbidity in longitudinal studies: Development and validation. Journal of
     Chronic Diseases, 40, 373 - 383">

  <SentryDefinition>
    <precondition processDefinitionId="MCS2_SelectPatient"/>
  </SentryDefinition>

  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.inclusiondiag"
    isReadOnly="false" isMandatory="true" position="LEFT"/>
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.ch1"
    isReadOnly="false" isMandatory="true" position="LEFT"/>
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.ch2"
    isReadOnly="false" isMandatory="true" position="LEFT"/>
  ...
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.bodysvg"
    isReadOnly="false" isMandatory="true" position="RIGHT"/>
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.ch18"
    isReadOnly="false" isMandatory="true" position="RIGHT"/>
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.ch19"
    isReadOnly="false" isMandatory="true" position="RIGHT"/>
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.ch20"
    isReadOnly="false" isMandatory="true" position="RIGHT"/>
  <TaskParamDefinition path="MCS2_Identification.MCS2_Charlson.ch21"
    isReadOnly="false" isMandatory="true" position="STRETCHED"/>

</HumanTaskDefinition>
```

DualTaskDefinition

A `DualTaskDefinition` declares a `DualTask` which represents a `DualTask` performed by a case-worker, followed by an `AutomatedTask` performed by an external system and is typically declared as part of a `StageDefinition`. The data layer binding is declared with `TaskParamDefinitions` that are part of a `DualTaskDefinition`. A `TaskParamDefinition` declares a `TaskParam` either as a `HUMAN` or an `AUTOMATED` part. The example below illustrates modeling a `DualTaskDefinition`, including `HttpHookDefinitions`, to synchronize information with third-party systems (cf. resulting screen in Figure 5.8). The dynamic description path enhances the task description dynamically depending on the selected attribute value.

Attribute	Required	Default	Description
`id`	mandatory	-	Declares the `id` used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the `id` is used as a technical human-readable `name` which is typically declared in English to enable debugging.
`description`	mandatory	-	Declares the `description` shown on the user interface and is typically expressed in the local language.
`ownerPath`	optional	-	Declares the path to the `DualTask` owner based on the `Case` root `Entity`. The path needs to be defined based on the `AttributeDefinition` names. Every path section is separated with a dot.
`dueDatePath`	optional	-	Declares the path to the `DualTask` due date based on the `Case` root `Entity`. The path requires to be defined based on the `AttributeDefinition` names. Every path section is separated with a dot.
`repeatable`	optional	`ONCE`	Declares the repeatability where `ONCE` allows only one execution, `SERIAL` allows multiple repetitions in serial order, and `PARALLEL` allows multiple active repetitions in parallel.
`isMandatory`	mandatory	-	Possible values are `TRUE` and `FALSE`.
`activation`	optional	`AUTOMATIC`	Declares the `activation` trigger which is either `MANUAL` by a case-worker, `AUTOMATIC` by the case execution engine, or depending on a declared `EXPRESSION`.
`manualActivation` `Expression`	optional	-	Declares the expression to be fulfilled for an `EXPRESSION`-based `activation`.
`entityDefinitionId`	optional	-	Declares which `EntityDefinition` is instantiated and attached as an `Entity` to the `entityAttachPath`.
`entityAttachPath`	optional	-	Declares the path where the newly instantiated `Entity` should be attached based on the `Case` root `Entity`. The path needs to be defined based on the `AttributeDefinition` names. Every path section is separated with a dot.
`externalId`	optional	-	Declares an `externalId` to allow mapping with an external system.
`dynamicDescription` `Path`	optional	-	Declares a path to an `Attribute` or `DerivedAttribute`, the value of which is used to extend the `DualTaskDefinition` `description` dynamically. The path specification depends on the `Case` root `Entity`. Every path section is separated with a dot.

XML Example:

```xml
<DualTaskDefinition
  id="MCS2_MonitoringPrescription"
  externalId="MonitoringPrescription"
  description="Monitoring Prescription"
  dynamicDescriptionPath="MCS2_Workplan.MCS2_MonitoringPrescription.type"
  ownerPath="MCS2_Settings.PrimaryCareClinicians"
  dueDatePath="MCS2_Settings.WorkplanDueDate"
  isMandatory="false"
  repeatable="PARALLEL"
  activation="MANUAL"
  entityDefinitionId="MCS2_MonitoringPrescription"
  entityAttachPath="MCS2_Workplan.MCS2_MonitoringPrescription">

  <HttpHookDefinition on="ACTIVATEAUTOMATEDPART"
    method="POST"
    url="http://integration-producer:8081/v1/sacm/prescription/monitoring"
    failureMessage="Could not create prescription on Self-Management System!"/>
  <HttpHookDefinition on="TERMINATE"
    method="POST"
    url="http://integration-producer:8081/v1/sacm/prescription/monitoring/terminate"
    failureMessage="Could not terminate prescription on Self-Management System!"/>

  <TaskParamDefinition part="HUMAN" path="MCS2_Workplan.MCS2_MonitoringPrescription.type"
    isReadOnly="false" isMandatory="true" position="LEFTCENTER"/>
  ...
  <TaskParamDefinition part="HUMAN" path="MCS2_Workplan.MCS2_MonitoringPrescription.startdate"
    isReadOnly="false" isMandatory="true" position="CENTERRIGHT"/>
  <TaskParamDefinition part="HUMAN" path="MCS2_Workplan.MCS2_MonitoringPrescription.enddate"
    isReadOnly="false" isMandatory="true" position="CENTERRIGHT"/>
  <TaskParamDefinition part="HUMAN" path="MCS2_Workplan.MCS2_MonitoringPrescription.timeslot"
    isReadOnly="false" isMandatory="true" position="CENTERRIGHT"/>
  <TaskParamDefinition part="HUMAN" path="MCS2_Workplan.MCS2_MonitoringPrescription.comments"
    isReadOnly="false" isMandatory="false" position="STRETCHED"/>
  <TaskParamDefinition part="AUTOMATED" path="MCS2_Workplan.MCS2_MonitoringPrescription.measurement"
    isReadOnly="false" isMandatory="true" position="STRETCHED"/>
</DualTaskDefinition>
```

■ AutomatedTaskDefinition

An `AutomatedTaskDefinition` declares an `AutomatedTask` which is performed by an external system and is typically declared as part of a `StageDefinition`. Compared to a `HumanTaskDefinition`, a due date is not declarable. Even though the execution is performed externally, an owner is declarable who is responsible for managing unforeseen `Alerts`. The data layer binding is declared with `TaskParamDefinitions` that are part of an `AutomatedTaskDefinition`. Modeling an `AutomatedTaskDefinition` is comparable with the `HumanTaskDefinition` except for the due date option.

Attribute	Required	Default	Description
id	mandatory	-	Declares the id used to resolve references within the XML declaration and is replaced with a dynamically generated id. After the import, the id is used as a technical human-readable name which is typically declared in English to enable debugging.
description	mandatory	-	Declares the description shown on the user interface and is typically expressed in the local language.
ownerPath	optional	-	Declares the path to the AutomatedTask owner based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
repeatable	optional	ONCE	Declares the repeatability where ONCE allows only one execution, SERIAL allows multiple repetitions in serial order, and PARALLEL allows multiple active repetitions in parallel.
isMandatory	mandatory	-	Possible values are TRUE and FALSE.
activation	optional	AUTOMATIC	Declares the activation trigger which is either MANUAL by a case-worker, AUTOMATIC by the case execution engine, or depending on a declared EXPRESSION.
manualActivation Expression	optional	-	Declares the expression to be fulfilled for an EXPRESSION-based activation.
entityDefinitionId	optional	-	Declares which EntityDefinition is instantiated and attached as an Entity to the entityAttachPath.
entityAttachPath	optional	-	Declares the path where the newly instantiated Entity should be attached based on the Case root Entity. The path needs to be defined based on the AttributeDefinition names. Every path section is separated with a dot.
externalId	optional	-	Declares an externalId to allow mapping with an external system.
dynamicDescription Path	optional	-	Declares a path to an Attribute or DerivedAttribute, the value of which is used to extend the AutomatedTaskDefinition description dynamically. The path specification depends on the Case root Entity. Every path section is separated with a dot.

TaskParamDefinition

Typically, multiple TaskParamDefinitions are used to declare the input and output data of a TaskDefinition. Therefore, a TaskParamDefinition specifies the linkage to an Attribute or DerivedAttribute on the data layer. An isMandatory flag indicates if a TaskParam must contain an AttributeValue to complete a Task or not. Depending on the declared multiplicity for the related AttributeDefinition, no AttributeValue might be compliant with a mandatory TaskParam when the multiplicity is declared as maximumOne or any. Examples are provided within the different TaskDefinitions.

Attribute	Required	Default	Description
path	mandatory	-	Declares the path depending on the Case root Entity, and points to the last existing path section.
isReadOnly	mandatory	-	Declares an isReadOnly boolean flag that expresses whether the TaskParam expects a valid input or is shown as read-only.
isMandatory	mandatory	-	Declares an isMandatory boolean flag that expresses whether the TaskParam must contain a valid value or might be empty when completing a Task.
position	optional	STRETCHED	Declares the position which enables declaring custom layouts for TaskDefinitions. Supported layout options are, LEFT, CENTER, RIGHT, STRETCHED over all three columns, LEFTCENTER stretchered over the first two columns, and CENTERRIGHT stretched over the last two columns.
part*	mandatory	-	DualTasks represent a HumanTask followed by an AutomatedTask. Therefore, each TaskParamDefinition must be declared as a HUMAN or an AUTOMATED part. The parameter is only applicable for DualTaskDefinitions.

*Only applicable for DualTaskDefinitions!

SentryDefinition

A SentryDefinition declares dependencies for ProcessDefinitions, that must be satisfied before the pending Process is enabled. Each SentryDefinition must have at least one precondition which refers a ProcessDefinition. Logically linked *AND* dependencies are expressible with multiple preconditions, whereas logically linked *OR* dependencies require declaring multiple SentryDefinitions. Additional dependencies regarding the data, might be declared with expressions. For simplicity reasons, the precondition is not specified further.

Attribute	Required	Description
expression	optional	Declares an expression that allows specifying dependencies on the data layer, but the evaluation is only performed after the preconditions are satisfied.

Default values are not supported!

XML Example:

```xml
<SentryDefinition expression="MCS2_Settings.Patient.age>65">
  <precondition processDefinitionId="MCS2_Identification"/>
  <precondition processDefinitionId="..."/>
</SentryDefinition>
```

HttpHookDefinition

A HttpHookDefinition is assignable to any ProcessDefinition. After a declared Process state transition, an HTTP request is performed to a specified URL, typically representing an external system. Considering when related services are deployed within a docker network, it is best practice to declare all URLs relatively depending on the docker network which allows using identical models for the development, test and production environment.

Attribute	Required	Description
on	mandatory	Declares the event triggering the hook execution. All Tasks support the following events: AVAILABLE, ENABLE, ACTIVATE, COMPLETE, TERMINATE, and CORRECT. Additionally, the DualTask supports the following events: ACTIVATEHUMANPART, ACTIVATEAUTOMATEDPART COMPLETEHUMANPART, COMPLETEAUTOMATEDPART, CORRECTHUMANPART, and CORRECTAUTOMATEDPART.
url	mandatory	Specifies the requested URL.
method	mandatory	Declares the request method with the supported values POST, GET, PUT, and DELETE.
failureMessage	optional	Declares a human-readable failureMessage to create a user-friendly error Alert when a hook execution fails.

Default values are not supported!

XML Example:

```xml
<HttpHookDefinition
  on="ACTIVATE"
  method="POST"
  url="http://integration/v1/prescription/monitoring/activate"
  failureMessage="Could not create prescription on Self-Management System!"/>

<HttpHookDefinition
  on="TERMINATE"
  method="POST"
  url="http://integration/v1/prescription/monitoring/terminate"
  failureMessage="Could not terminate prescription on Self-Management System!"/>
```

Action

The test declaration enables reliable, repeatable testing that is easily executable after template adaptions and therefore helps to increases the case template maturity. Several Actions enable a test flow declaration for a case template. Typically, Actions are declared that could be performed on the user interface or simulate the interaction with a third-party system. Breakpoints are declarable on each Action to debug complex models step by step. A specified breakpoint stops the execution before the Action while pressing enter on the console resumes to the execution flow. The debug functionality currently requires command-line access and is therefore limited.

Attribute	Required	Description
type	mandatory	Declares the type which should be performed. Currently, the task-centered Actions such as ActivateStage, CompleteStage, ActivateHumanTask, ActivateDualTask, CompleteHumanTask, CompleteAutomatedTask, CompleteDualTaskHumanPart, CompleteDualTaskAutomatedPart, CorrectHumanTask, and CorrectDualTaskHumanPart are supported. Additionally, a custom Alert can be created with the type CreateAlert and execution delay can be specified with the type Delay.
processDefinitionId	mandatory	Declares the processDefinitionId referring to the ProcessDefinition which is automatically resolved to the related Process during test execution.
creationDate*	optional	Declares an creationDate for CreateAlert Actions.
expireDate*	optional	Declares a expireDate for CreateAlert Actions.
text*	optional	Declares a text for CreateAlert Actions.
data*	optional	Declares additional JSON data for CreateAlert Actions.
breakpoint	optional	Declares a breakpoint that interrupts the test execution for debugging purposes. Possible values are TRUE and FALSE. A FALSE value leads to a similar execution semantic as removing the breakpoint.

*Only applicable for CreateAlert Actions. Default values are not supported!

TaskParam

All Actions that modify Task data must declare a TaskParam within those Actions. A TaskParam references the data similarly with a path as the TaskParam declared within the TaskDefinition. The linked Attribute values must be set to complete a Task. Therefore, a values attribute enables declaring all crucial AttributeValues.

Attribute	Required	Description
`path`	mandatory	Declares the `path` for data binding. The `paths` are used to map the `Task` `TaskParam` with the aid of the related `TaskParamDefinition`.
`values`	mandatory*	Declares an array of `values`, whereas the declared linked `AttributeDefinition` and the related constraint such as the multiplicity must be considered to provide valid `values`.
`userValue`	mandatory*	Declares a `userValue`, which is a shortcut for declaring a link to an existing user that supports using the human-readable XML `User` id. Alternatively, within the `values`, a JSON-like object containing the persisted `Users` id can be declared. E.g., `'{id:363d54sd75d4376}'`

*Either the `values` or the `userValue` attribute must be declared, comparable to a logical *XOR*. Default values are not supported!

XML Example:

```
<Execution>

  <!-- Case Identification Sample -->
  <Action type="CompleteHumanTask" processDefinitionId="MCS2_AssignRoles">
    <TaskParam path="MCS2_Settings.Patient"  userValue="HopkinsC"/>
    <TaskParam path="MCS2_Settings.Clinician"  userValue="HamiltonA"/>
    <TaskParam path="MCS2_Settings.Nurse"  userValue="HuntL"/>
  </Action>
  <Action type="CompleteHumanTask" processDefinitionId="MCS2_Charlson">
    <TaskParam path="MCS2_Identification.MCS2_Charlson.inclusiondiag"  values="['LKP']"/>
    <TaskParam path="MCS2_Identification.MCS2_Charlson.ch1"  values="['0']"/>
    <TaskParam path="MCS2_Identification.MCS2_Charlson.ch2"  values="['1']"/>
    <TaskParam path="MCS2_Identification.MCS2_Charlson.ch3"  values="['0']"/>
    ...
  </Action>

  <!-- Workplan Sample -->
  <Action type="ActivateDualTask" processDefinitionId="MCS2_Monitoring"/>
  <Action type="CompleteDualTaskHumanPart" processId="MCS2_Monitoring">
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.type"  values="['BLOODPRESSURE']"/>
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.alertmin"  values="[90, 60]"/>
    <TaskParam path="MCS2_Workplan.MCS2_Monitoringn.alertmax"  values="[180, 100]"/>
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.comments"  values="['Take your time!']"/>
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.startdate"  values="['2019-05-09T00:00:00.0']"/>
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.enddate"  values="['2019-05-12T00:00:00.0']"/>
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.timeslot"  values="['BREAKFAST', 'DINNER']"/>
  </Action>
  <Action type="CreateAlert" processDefinitionId="MCS2_Monitoring" text="Patient did not accomplish
     measurement as planned!"/>
  <Action type="CompleteDualTaskAutomatedPart" processDefinitionId="MCS2_Monitoring">
    <TaskParam path="MCS2_Workplan.MCS2_Monitoring.measurement"  values="[
      {"Systolic": [
        {"date": "2019-05-11 10:03:53.0", "value": 95},
        {"date": "2019-05-11 18:54:53.0", "value": 130}
      ]},
      {"Diastolic": [
        {"date": "2019-05-11 10:03:53.0", "value": 73},
        {"date": "2019-05-11 18:54:53.0", "value": 96}
      ]}
    ]"/>
  </Action>

</Execution>
```

5.3. Case Modeling Best Practice Principles

Modeling case templates is an emergent knowledge-intensive process. Different domain-specific challenging issues occur. However, the solutions indicate similar principles. We used our modeling experience from several sophisticated integrated care use cases in different organizations to derive case modeling best practice principles, which are:

1. **Use Consistently End-Users Terminology** Domain-specific terminology emerges over time and ensures a common language for communication. The end-user usability increases significantly when domain-specific terminology is used consistently for modeling the case template. Therefore, a case-modeler with domain-knowledge helps express domain-specific terminology precisely within the model.

2. **Start with Less Restrictive Models and Enhance** When starting modeling, the stakeholders work closely with the case template modeler to share their knowledge. Stakeholders are mostly domain experts and typically explain their default processes considering all common restrictions which might not be applicable to all cases. The modeler must keep the balance between general restrictions guiding the case-worker and the degree of flexibility needed to support case-workers in accomplishing their work. Considering that knowledge-intensive work is not predictable, the best approach is to start with a minimal set of restrictions which are actually required and to add more restrictions if needed based on the experience from the first case executions.

3. **Setup Iterative Feedback Loops with Stakeholders** Modeling is a complex endeavor and needs use case-specific knowledge. We noticed that communicating the difference between hard-wired system-based features and dynamical adaptable model-based features to non-technical stakeholders is challenging. Stakeholders model requirements are often vaguely described and need several refinements for a precise description. Typically, a conceptualization is missing to enable modeling. Practically, regular iterative feedback loops help create a common understanding. Short adaption cycles lead to an emerging case template that continuously improves. Stakeholders can execute the improved case template immediately and provide practical feedback rather than theoretically describing occurring issues. Supposed small model-based changes often significantly improve the daily work of a case-worker. Once the case template is usable, stakeholders often use gain knowledge to conceptualize similar issues.

4. **Reuse Existing Modeling Patterns** Many stakeholder requirements seem to be specific and unique. However, conceptualization helps to identify their abstract needs which might be coverable with existing modeling patterns. Obviously, the declared flow is customizable within a model-based adaptive case management system. However, to satisfy end-user needs and to ensure the desired usability, more customization is required. We noticed several modeling patterns which are reused multiple times. In our context, `Tasks` mostly represent clinical questionnaires that result in a dynamically calculated numeric score. Coloring the numeric score depending on result thresholds as a traffic light with `CustomDataRepresentations` helps to explain numeric results implicitly. Additionally, with a `DerivedAttributeDefinition`, the numeric result can be transformed into textual representation to describe the result, or alternatively, an `additionalDescriptions`

might be used to provide explanatory information. Several questionnaires represent knowledge that can be presented visually attractive as dynamically generated graphics with a `CustomDataRepresentations`. Depending on the use case, individual dynamically calculated SVG graphics depending on attribute values might be modeled. Each case template allows defining `SummarySections` that represent a dynamically generated patient cover sheet. The notes feature might be used simply for unstructured documentation or is customized to a specific use case. Declaring an HTML template allows structuring the unstructured documentation approach. An HTML template we noticed is used to create a role-specific communication wall. The first column represents the clinical professionals and the second column, the primary care professionals, while each row represents a dedicated row. The application of existing modeling patterns will cover many specific requirements.

5. **Declare Environment Independent Models** Ordinarily, software services are deployed in multiple environments to ensure a separation of concerns. Software developers use a dedicated development environment to test the latest builds, integration tests are applied in a test environment, and a stable production environment provides the service for end-users. Before new case templates become available for end-users, they should be tested in the test environment to ensure a certain maturity. Therefore, from a design perspective, the identical case template should be executable without any change in all environments. However, the integration with third-party services is typically modeled with hooks that request a declared URL. Considering the related services are all deployed within a docker network, the URLs declared within the model should refer to the relative docker network instead of a hard-coded domain. The similar pattern should be applied for declarations containing links. Following this approach enables using identical models in all environments.

6. **Declare Executable Tests** The configuration complexity of individual model elements seems manageable. With each additional model element used to declare a case template, the entire complexity increases. The emerging template complexity becomes challenging to control, considering the interactions and constraints between all model elements. Modeling is a complex endeavor where simple model changes may create unforeseen side effects. Ideally, case template adaptations are tested manually by the modeler with a clear focus on the applied change. Typically, case templates are too extensive to manually perform integral testing. Therefore, declaring an executable test flow within the case template helps to significantly increase the model maturity. Issues are detected more systematically and faster.

7. **Use a Model-Based Integration Approach** Providing an integrated information perspective is crucial for knowledge-intensive processes. Today, needed information is typically managed distributed across several specialized information systems. The orchestration and synchronization with third-party systems which are mostly non-model-based are challenging. Developing individual integration concepts is time-consuming and leads to incoherent solutions that require additional maintenance effort. Therefore, our conceptual design provides specific model elements supporting the integration and orchestration of external systems. A third-party process step is modeled with an `AutomatedTask`, while the `HttpHookDefinitions` concept enables notifying external systems on a specific state change. The `DualTask` is specially designed for integrating a collaborative task and con-

sists of a HumanTask accomplished by a case-worker and is followed by an AutomatedTask performed by an external system. Unforeseen domain-specific exceptions are representable with custom Alerts assigned to any Process element. Multiple model elements support declaring externalIds to simplify mapping primary identifiers of external systems. On the instance level, instance-specific identifiers are supported. Additionally, the User and Group management supports using custom primary identifiers if the uniqueness is ensured. Using existing concepts also reduces the communication and documentation effort. Therefore, we recommend re-using the existing integration patterns before considering a hard-wired integration.

5.4. Model Import

Declaring a single model element is rather simple and possible with an API request. Modeling a holistic case template is rather complex. The combinatorial complexity increases with each additional element. Defining a holistic case template manually on the API level is theoretically possible but not practicable. Therefore, the case templates are declared using XML that can be imported. Section 5.4.1 illustrates the conceptual workspace import steps and Section 5.4.2 illustrates the conceptual case templates import steps.

5.4.1. Workspace Import Steps

The workspace import allows initializing a newly deployed instance. First, an XML file is parsed, role-based elements are created, then the workspace is imported. The detailed steps are follows:

1. **Load Workspace File** The XML workspace file is parsed with a sax parser and transformed into a large JSON-based tree. The importer translates the XML definition into API requests which are created or update the desired model elements.
2. **Initialize Mappings** Declared XML elements are mutually dependent. Therefore, XML references must be mapped with the created resource identifies. When an XML element refers to another element, the mapping is used to resolve the XML reference to the actual existing resource identifier. Each newly imported element extends the mapping.
3. **Update UserDefinition** The default user definition is updated, including their declared attributes and derived definitions.
4. **Create Users** All declared users are imported with their declared attribute values. The number of attributes and the types of attributes depend on the user definition.
5. **Create Groups** Each group must declare an administrator of the type principle, which is either a user or a group. During the import, groups are ordered according to their dependencies to ensure that a group reference is resolvable. Cyclic dependencies are not supported.

6. **Create Memberships** After the groups are imported, the related memberships are created. First, the memberships of all imported groups and then the system administrator memberships are created.
7. **Update Settings** The global system settings are updated to grant privileges such as creating a workspace, group, or user to certain principles.
8. **Create Workspaces** Finally, the workspace with the declared administrator principle, writer principle, reader principle, and contributor principle are imported.

The import helps to recreate a clean test or development environment quickly. After fundamental changes in the execution engine or on the persistence layer, the workspace XML file is adapted if needed, the existing development or test database is deleted, and a re-import is triggered. This approach reduces unintentional side effects and increases the reproducibility of the results. The import is similarly valuable for the initial setup of a production environment. This approach enables starting with a tested configuration setting. However, later modifications such as adding an additional workspace which might be needed on a production environment are not supported yet. Therefore, these adaptions must be applied directly on the API level.

5.4.2. Case Template Import Steps

The case template import flow enables adding new templates to an existing workspace. First, an XML case template file is parsed, the data schema is created, then the process definitions are created. The detailed steps are as follows:

1. **Load Case Template File** The XML case template file is parsed with a SAX parser and transformed into a large JSON-based tree. The API-based importer enables importing XML templates which are contained in the deployment or attached as payload to the import request.
2. **Initialize Mappings** Several case definition modeling elements are pending on common instance elements such as workspace, group, and user as declared in Section 5.4.1. Therefore, the importer needs to resolve the common modeling elements to actual existing identifiers of instances from the database. All common elements allow assigning static identifiers which can then be bound to the XML reference element. During the mapping initialization, the local XML identifier is mapped to the static instance identifier for all workspaces, groups, and users. With every following import step, the mapping is extended, thus allowing to reference the newly created elements as well.
3. **Create EntityDefinitions** First, the mapping is used to resolve the parent workspace into an existing instance identifier. Then all entity definitions are created subsequently.
4. **Create AttributeDefinitions** For every entity definition, all related attribute definitions with all declared constraints are created, which includes attribute definitions referencing other entity definitions.
5. **Create DerivedAttributeDefinitions** Derived attribute definitions mostly reference attribute definitions and are therefore imported after the attribute definitions.

6. **Create CaseDefinitions** After all data definitions are imported, the process definitions follow, beginning with the case definition. The case definition is attached to the parent workspace resolved via mapping.
7. **Create SummarySectionDefinitions** All case definition related summary sections are created, including the summary section parameter definitions that reference to the created attributes.
8. **Create StageDefinitions** All Stage definitions contained within case definition in are created recursively, beginning with the first root stage definition.
9. **Create TaskDefinitions** All task definitions are created beginning on the case definition root level, then recursively for all stage definitions. During the creation of a human task definition, dual task definition, or automated task definition related task parameters are created as well. After creating a task definition, the related hook definitions are created.
10. **Create SentryDefinitions** All Sentry definitions create declaring preconditions across all process definition elements, such as stage definitions, human task definitions, dual task definitions, and automated task definitions.
11. **Mark as Instantiable** When creating a case definition, the flag is instantiable is set by default to false, thus preventing the instantiation of the case during the import. This step marks the case definition as instantiable and prevents instantiating previous case definition versions by setting the flags of those to false.
12. **Execute Declared Tests** The XML structure allows defining a sample case execution flow. The execution needs to know the actual ids of the created models and therefore, it can only be executed directly after an import is performed. Testing is only desired when importing to a non-productive environment and therefore, it can be enabled and disabled with flags on the import endpoint.

The case template import is typically used on the development, test, and production environments. Versioning enables importing the similar case template multiple times by increasing the version number parameter. For production usage, small incremental changes are typically uploaded as a new case template after they are tested on a test environment. Theoretically, the API allows the modification of existing case templates, but this is rather complex due to possibly occurring side effects of already instanced cases and is not desired.

5.5. Conceptual API Design

In the following, the conceptual API design aspects are elaborated. In Section 5.5.1, fundamental API design principles are described, in Section 5.5.2, different API authentication headers are briefly explained, and in Section 5.5.3, the response structure is exemplarily illustrated based on a `HumanTask` resource.

5.5.1. API Design Principles

The conceptual API design uses the REST design principles as a reference and deviates where required. REST is an abbreviation for representational state transfer and represents an architectural style initially developed by Fielding (2000). Fielding determined four primary aspects of successful world wide web applications as the foundation for designing REST and derived six restrictions, namely performance, scalability, simplicity, modifiability, portability, and visibility which form the fundamentals of modern RESTful API interfaces. Additionally, best practices for designing a RESTful API are considered where applicable as presented by Gebhart et al. (2015a,b).

Basically, meta-model elements are mapped to API resources. Each resource URL contains a version number that allows modifying or extending a resource and publishing the changes without breaking any existing client applications. The convention to represent all resources with plural nouns is considered. The resource naming follows the specific terminology used within the meta-model as far as possible. Content is typically expressed with the JavaScript Object Notation (JSON) and therefore, all attributes follow the camel-case JavaScript naming convention.

The client-server communication is stateless and preferably uses JWT tokens to authenticate users. All requests are UTF-8 encoded and use the content type application/json where possible. A gzip compression is used to significantly reduce the response size and the bandwidth required for transmission. Where required, query parameters enable pagination to limit the results. Pagination ensures a decent response time with a growing amount of data.

Errors are expressed with HTTP-specific status codes. A meta-model-based API implicitly has an additional abstraction layer to support modeling which increases the complexity to debug occurring errors. Therefore, providing human-interpretable error messages where possible is crucial. Detailed technical error messages might increase the potential for security issues and must be well thought. The API design focuses on providing detailed meta-model validation errors on all endpoints to increase usability.

As HTTP methods, we use POST to create new resource instances, GET to return existing resource instances, PATCH to allow partial resource instance updates and DELETE to remove a resource instance. The complexity of specific resources is comparably high. Therefore, we decided to use the HTTP PATCH method for partial updates instead of the HTTP PUT method to set the overall object. Our design typically allows simple REST API requests on the model definition level whereas on the instance level, mostly function-based requests are used to encapsulate the complexity of the case execution engine.

In addition to the meta-model-based validation, which is applied on each modifying request, the explicitly declared and implicitly inherited access rights are considered. A certain user might have read access to a resource but no write access. A primary design consideration is to encapsulate complex logic on the execution engine to reduce the complexity of the client application. Therefore, the case execution engine derives a list of possible actions considering the current state of resources and the user's access level for non-trivial actions such as completing a task, changing the case owner, and similar actions. Those possible actions are returned embedded within resources.

The meta-model indicates the strong relationship between concepts. Typically, a client application must know contextual information that the resource links. I.e., a task references an owner and when loading a task resource, the related owner resource is needed for processing as well. Therefore, the endpoints typically embed related resources simplified to reduce the number of round trips when needed. Regardless whether a JSON attribute is represented with a simplified resource or merely with a resource id, the JSON attribute name does not change to provide an equivalent data structure for all related requests. This pattern is comparable with the plural resource names even if only maximum one resource is expected as a result.

Multiple concepts are inherited within the meta-model, while all inherited concepts support unique ids across all resources that allow requesting a `Stage`, `HumanTask`, `DualTask`, or `AutomatedTask` resource based on the `Process` resource endpoint. This pattern increases resource re-usage. However, modification operations are typically only directly supported on the dedicated resource endpoints, thus preventing unintended operations.

The endpoint documentation is generated based on annotations declared for each endpoint. Additionally, a postman[2] collection is maintained for development and testing purposes. A detailed `HumanTask` sample response is presented in Section 5.5.3, and a high-level API reference lists essential endpoints in Section A.2.

5.5.2. Authentication Headers

The ACM4IC API allows authentication with three different methods. In a production context, the usage of JSON Web Tokens (JWT)[3] is recommended. The Hybrid Wiki legacy authentication method basic auth can be used, but is not recommended. For development purposes, the authentication might be simulated using the user's registered email address. In the following, the request headers are described in detail:

Option 1: JSON Web Token (JWT) Header

```
Content-Type: application/json
Authorization: Bearer <JWT>
```

Hint: Inconvenient for development purposes when expire times are short!

Option 2: Basic Auth Header

```
Content-Type: application/json
Authorization: Basic <<email>:<password>>
```

Hint: The email-password string must be base-64 encoded!

Option 3: Simulate-User Header

```
Content-Type: application/json
Simulate-User: mustermann@test.sc
```

Hint: Only applicable for development mode!

[2] https://www.getpostman.com, accessed on January 8, 2019

[3] https://tools.ietf.org/html/rfc7519, accessed on January 8, 2019

5.5.3. Response Structure

This section illustrates the detailed GET response for a `HumanTask` resource. The yellow colored headline presenting the HTTPS Method and the URL indicate that the resource is part of the case execution layer. The response headers declare the crucial content-type application/json and declare the charset UTF-8 encoding. Additionally, for content compression, gzip is applied that reduces the payload size. Figure 5.17, represents a JSON serialized response body of a `HumanTask` while specific attributes are not illustrated for reason of simplification reasons. Most meta-model elements have a strong relationship with other meta-model elements. The API design is a depiction of the meta-model (cf. Section 4.2). Therefore, the strong relationship is visible similarly on the resource responses. To illustrate those relationships, the colored rectangle in front of each line indicates the architectural layer (cf. Section 4.1).

Each resource has an `id` and a `resourceType`, which uniquely addresses a resource. The `HumanTask` instance primarily depends on the `HumanTaskDefinition`, which is accessible with the JSON attribute `processDefinition` colored in purple. For reason of simplification and usability, all model definition attributes are injected directly at the root level, which includes the `description`, `name`, `isMandatory`, `repeatable`, `isManualActivation`, and `footnote` attribute. The JSON attribute, `isManualActivation`, is dynamically derived from the `activation` attribute from the `ProcessDefinition`. Besides the related model definition injected on the root level, all further contextual resources are embedded as a simplified nested JSON object or merely referenced with the resource `id`. Each `HumanTask` belongs to exactly one `case`. Within the detailed task endpoint, the JSON attribute `Case` only serializes the referenced `id`. In the context of an endpoint returning all user's active `HumanTasks` across all `Cases`, which is used for the dashboard page, the simple `id` is replaced with a nested object containing the `id` and the related `Case` name which is relevant for the dashboard.

Each `HumanTask` has a JSON attribute `owner` that is represented with a simplified nested `User` object colored in red. Additionally, the related `ownerConstraint` is represented by an array of simplified nested `Groups` which indicate the role constraint. A `mayEdit` flag simply indicates whether the current user has read or write access. The `possibleActions` array lists all currently executable actions considering the user's access level, case roles, and the state of the resource. A `HumanTask` typically has multiple `taskParams` that are needed to represent a task details page. The `TaskParam` links to the data layer and refers either to an `Attribute` or `DerivedAttribute`. Due to the complexity, the `Attribute` indicated in dark-blue and the `AttributeDefinition` indicated in light-blue, are embedded on the `TaskParam` root level. Additionally, the flags `isMandatory` and `isReadOnly` are merged from the `TaskParamDefinition` colored in purple. For the user interface representation, the layout `position` attribute and `uiReference` attribute declaring a `CustomDataRepresentation` are contained and colored in gray.

The endpoint-specific embedding of objects helps to provide a balance between API usability for client applications and consistency considerations. I.e., all endpoints returning multiple `Tasks` do not embed the `TaskParams` because they are only needed for a detailed representation. In general, most resources provide three abstraction levels such as: i) embedded objects naturally serializes only most relevant meta information, ii) an endpoint that returns a list with multiple resources typically serializes merely all meta information, and iii) an endpoint that returns maximum one resource serializes all meta information and includes all details.

```
GET                https://<host>/api/v1/humantasks/b6h6zo48dwi1
Content-Type:      application/json; charset=utf-8;
Content-Encoding:  gzip
Body:
🟨 {
🟨   "id": "b6h6zo48dwi1",
🟨   "resourceType": "humantasks",
🟦   "workspace": "2c9480885d1737ef015d74deed260006",
🟨   "case": "d5rbxu0ivx30",
🟪   "processDefinition": "k4x10x3mp2bc",
🟨   "parentStage": "1hsxgvgp8uzex",
🟨   "state": "COMPLETED",
🟨   "stateTransitions": {
🟨     "AVAILABLE": {...}
🟨     "ENABLED": {...},
🟨     "ACTIVE": {...},
🟨     "COMPLETED": {
🟨       "by": {
🟥         "id": "2c9480845bee03e7015bfcad28990010",
🟥         "email": "sophie.werner@xxx.de",
🟥         "name": "Sophie Werner",
🟥         "resourceType": "users"
🟨       },
🟨       "date": "2018-11-22 16:42:53.0"
🟨     },
🟨     "TERMINATED": {...},
🟨   },
🟨   "possibleActions": [ "CORRECT" ],
🟨   "isHighlighted": false,
🟨   "nrLogs": 0,
🟨   "nrAlerts": 0,
🟨   "nrAlertsUnseen": 0,
🟨   "alerts": [...],
🟪   "description": "Charlson",
🟪   "name": "GCS2_Charlson",
🟪   "isMandatory": true,
🟪   "repeatable": "ONCE",
🟪   "externalId": null,
🟪   "isManualActivation": false,
🟨   "dueDate": "2018-11-26 00:00:00.0",
🟨   "isOverdue": false,
🟥   "ownerConstraint": [{
🟥     "id": "2c9480885d1737ef015d74deed260006",
🟥     "name": "Clinican",
🟥     "resourceType": "groups"
🟥   }],
🟥   "owner": {
🟥     "id": "2c9480845bee03e7015bfcad28990010",
🟥     "email": "sophie.werner@xxx.de",
🟥     "name": "Sophie Werner"
🟥     "resourceType": "users"
🟥   },
🟥   "mayEdit": true
🟨   "index": 0,
🟨   "prev": null,
🟨   "next": null,
🟨   "isVisibleOnDashboard": false,
🟨   "taskParams": [
🟨     {
🟨       "id": "16zbqd8qvv34i",
🟨       "resourceType": "taskparams",
🟪       "isMandatory": true,
🟪       "isReadOnly": false,
🟦       "name": "ch2",
🟦       "description": "2. Congestive heart failure",
🟦       "additionalDescription": null,
🟦       "isDerived": false,
🟦       "multiplicity": "exactlyOne",
🟦       "attributeType": "enumeration",
🟦       "externalId": null,
🟦       "attributeTypeConstraints": {
🟦         "enumerationOptions": [
🟦           {
🟦             "value": "0",
🟦             "description": "No",
🟦             "additionalDescription": null,
🟦             "externalId": null
🟦           },
🟦           {
🟦             "value": "1",
🟦             "description": "Yes",
🟦             "additionalDescription": null,
🟦             "externalId": null
🟦           }
🟦         ]
🟦       },
🟦       "defaultValues": [],
⬛       "values": [ "1" ],
◼       "position": "STRETCHED",
◼       "uiReference": null
🟨     },
🟨     ...
🟨     {
🟨       "id": "sxgoaswpetfh",
🟨       "resourceType": "taskparams",
🟪       "isMandatory": false,
🟪       "isReadOnly": true,
🟩       "name": "ch21",
🟩       "description": "Charlson",
🟩       "additionalDescription": null,
🟩       "isDerived": true,
🟩       "evaluationError": null,
🟩       "attributeType": "number",
🟩       "externalId": null,
🟩       "values": [ 17 ],
◼       "position": "STRETCHED",
◼       "uiReference": "colors(1<=green<=2<orange<=4<red<100)"
🟨     }
🟨   ],
🟪   "footnote": "Mary E. Charlson, Peter Pompei, Kathy L. Ales & C.Ronald MacKenzie (1987). A new
     method of classifying prognostic comorbidity in longitudinal studies: Development and validation.
     Journal of Chronic Diseases, 40, 373-383"

🟨 }
```

Figure 5.17.: Example `HumanTask` resource response structure.

5.6. Technical Challenges

During the prototypical implementation, a variety of technical challenges occurred, while this section highlights the crucial technical challenges. In Section 5.6.1, the complexity of modifying dedicated case access rights is illustrated. Challenges regarding the type-safe queries based on the existing ORM-Engine are presented in Section 5.6.2 and the serialization of complex aggregated objects in combination with the existing ORM-Engine is described in Section 5.6.3. Challenges regarding system changes and their interrelation with models are briefly explained in Section 5.6.4.

5.6.1. Complexity of Modifying Case Access Rights

The underlying Hybrid Wiki architecture supports managing access rights on the workspace or on the instantiated object level. Instantiated cases either have inherited access rights from a workspace or dedicated case access rights. Typically, the administration access rights are inherited from the corresponding workspace and all other access rights are dedicated to the specific case instance. Supported case access levels are read, write, and case owner. Adding or removing case-specific access rights would enforce the persistence layer to update all case-related objects with the new access rights. A typical case has a significant number of processes, related entities, alerts, and messages that are usually increasing until the case is completed or terminated. During the case instantiation, write access is granted to the current session user. Task parameters used to set a task owner grant write access for the selected user if that user does not have write access yet. After a case has been instantiated, most templates contain a task that assigns users to all roles of a case. Granting or revoking access rights is implemented with a listener concept on a task parameter change. The fact that each role change might trigger refreshing the access rights in combination with the growing number of objects, the default Hybrid Wiki access rights architecture does not scale well. Therefore, instead of assigning an access token for each user to all case-related objects, a case writer group and a case reader group are used, thus improving the performance significantly. Once the first write access should be granted, a case writer group is created, assigned to all case objects, and the user becomes a member of this group. Granting write access for a second user is accomplished by only adding a group membership instead of updating all case objects. A similar pattern is applied for the case readers. The technical case groups are only used for performance optimization and they are not visible on any API.

5.6.2. Type-Safe Queries Based on the ORM Engine

Hybrid Wiki uses a self implemented object-relational mapping that supports creating type safe SQL queries based on classes inherited from persistent entities, comparable with the Java criteria API in combination with the meta-model API[4]. For most use cases, the expressiveness is sufficient. However, certain queries need specialized *joins* or the *as* operator to distinguish tables that are contained multiple times within a query. We noticed the issue on the following use case. For a dashboard, all unread case related messages for an individual user and workspace depending on their the access rights should be shown. A message is associated with a case and those with the workspace. Every case reader might mark case messages individually as read. During

[4] https://docs.oracle.com/javaee/6/tutorial/doc/gjivm.html, accessed on February 27, 2019

run-time, the case readers may change dynamically. A typical implementation would store the information regarding who has read a message normalized within an additional relation to solve the issue that a user has M messages and N Users can read each Message. The query below illustrates conceptual how to simply fetch all unread messages with a native SQL *right join*:

```
SELECT message
FROM message_user mu RIGHT JOIN message m ON mu.messsage = m.id
WHERE mu.id IS NULL
```

Alternatively, a query with an SQL *as* operator could be defined to first join all read messages within a subquery and then to select only messages that are not contained within the subquery. Apart from that, the performance is not optimal, the *as* operator is not supported by the ORM. Conceptually, integrating the information who read a message into the message itself leads to a feasible approach. Therefore, the message contains a string attribute that concatenates the user ids and uses an underscore as a delimiter. Persistent attributes are repented with a hierarchy of classes distinguishing different attribute types, including string, date, number, and several more. To encapsulate the complexity, the existing *StringProperty* class is extended to support adding or removing a reader with conventional methods. For each operation the string is split, converted into a set, the operation is performed, and transformed back to the concatenated string representation. The following query is created with native ORM capabilities:

```
Query q = myCasesByReadAccessQuery();
q.addJoin(new Join(Workspace.SCHEMA.prototype().id, Case.SCHEMA.prototype().workspace.getAttributeSign()));
q = new QueryAnd(q, new QueryEquals(case.SCHEMA.prototype().workspace, "178kv3cy2d99v"));
q.addJoin(new Join(Case.SCHEMA.prototype().id, Message.SCHEMA.prototype().case.getAttributeSign()));
q = new QueryAnd(q, new QueryNot(new QueryContains(Message.SCHEMA.prototype().seen, "1a37obyp3gs12")));
Message.SCHEMA.queryEntities(q);
```

Each user has an access token which contains all group ids serialized that are accessible by that user. All case-related objects have a read access attribute that materializes serialized as a string the identifiers of the readers, who can access the element. The serialized identifiers are separated with a unique character that must not be contained within a valid identifier. Users or groups with write access are also contained within the materialization to provide adequate query performance. In the first step, the users access tokens are de-serialized and transformed into a *or* concatenated query to consider access rights. Secondly, the workspace and case schemata are joined and a workspace filter is applied. Thirdly, the case and message schemata are joined and a filter is applied that only unread messages are contained. Therefore, the seen attribute contains the concatenated users who have previously read the message. The ORM translates the meta-model-based query into a native executable query that is performed directly in a relational database:

```
SELECT message
FROM message JOIN case ON message.case = case.id JOIN workspace ON case.workspace = workspace.id
WHERE ( case.readAccess LIKE ’%10535812mq79k%’ OR case.readAccess LIKE ’%12mnu5216n7v1%’ ) AND
workspace.id = ’%178kv3cy2d99v%’ AND NOT ( message.seen LIKE ’%1a37obyp3gs12%’ )
```

The ORM transforms a *QueryContains* object into an SQL-*like* operator with a pre- and post-search term wildcard. Conceptually, this might be suboptimal. However, considering index structures, the solution is applicable for our use case. The pattern of serializing user ids is applied multiple times to prevent creating an access rights table for each meta-model element which allows declaring access rights. Clearly, native SQL queries could be used directly to query the data. However, this would violate the architectural design, prevent creating reusable methods, prevent using existing serializers, and complicate future refactoring.

5.6.3. Serialization of Complex Aggregated Objects with the Existing ORM

The existing ORM framework allows defining custom meta-model-based queries that are transformed into native executable SQL statements. However, as return type only defined meta-model elements are declarable. Considering that specific API endpoints require aggregation information, the default serialization process is not applicable anymore. The my-case page lists all cases considering the access rights of the currently logged in user. The visible meta-information for each case can be distinguished into static information and dynamically aggregated information. All static information such as case name, state, patient, case owner is directly accessible from the case object, whereas the aggregated information requires dynamical user-dependent computation. For each case the number of non-acknowledged notifications and the number of personnel non-acknowledged notification is visible. Similarly, the number of pending tasks and the personal pending tasks for the current user are shown. Additionally, the number of personal unread messages is indicated.

The described issue occurred on multiple API endpoints. Therefore, a generic architectural pattern is designed to handle those challenges. The default queries expressively support implicit access right checking and other valuable functionalities. To provide an architectural compliant approach, the generic JSON serializer class is extended with an additional method that is executed before the serialization is performed to enhance serialization items. Within the meta-model element-specific subclass, the generic enhancement method is overridden to provide an element-specific enhancement. The method expects a generic iterable as input parameter and a modified generic iterable as return type. The expected aggregated values which should be enhanced must be declared as non-persistent attributes within the model. Only one additional database round-trip is required to enhance the serialization of a complete list independent of their length. Compared to the default implementation with the ORM where the enhancement of each list item causes an additional round-trip, this approach is a significant performance improvement.

5.6.4. Change Management

The evolutionarily evolving system design has implications on the technical implementation and possibly on instantiated case models. Adding support for new independent model elements is simple. Considering that capabilities of existing model elements must be extended, specific issues arise. I.e., initially, repetitions have been modeled with a boolean flag. When introducing the modeling capability to distinguish between serial and parallel repetitions, a triple was required. Therefore, the boolean flag representing a `TRUE` and `FALSE` value is replaced with an enumeration supporting the values `ONCE`, `SERIAL` and `PARALLEL`. In addition to the database migration, all case template declarations must be migrated to prevent inconsistency issues supporting legacy notations. Changes of this category are still comparably simple to migrate because the migration rule is static for all instantiated cases and case templates.

The meta-model-based integration strategy provides many benefits, such as the re-usability of integration modeling patterns. However, tough challenges typically occur on the integration interfaces where the break between meta-model-based and non-meta-model-based systems must be considered. Hooks are frequently used to notify third-party systems on a specific event with a request on a URL declared within a meta-model. When the service URL changes, the case template must be adapted, as well as the already instantiated cases containing these hooks. A more complex migration rule will be able to handle the database migration. Similar migration patterns may be applied when the capabilities of an existing third-party service are extended, and the returned data format is changed so it is incompatible with the existing data format. Alternatively, the endpoints of the external system are versioned and only the case template is adapted to use the new version. Considering that changes are required, the case execution engine will also apply those changes to the already existing cases, and external services must support changes for existing and new case. All types of changes are manageable but a high communication effort is required to synchronize those changes across system boundaries.

5.7. Summary of Prototypically Supported Requirements

The requirements have been derived from the literature (cf. Chapter 3) and the conceptual implementation of the requirements has been elaborated (cf. Chapter 4). This chapter details the prototypical implementation of the requirements. Table 5.1 lists end-user interface features as rows and the requirements as columns. The resulting matrix illustrates which end-user interface feature supports a requirement. A requirement is considered as partly supported if the requirement is not fully satisfied by a end-user interface feature. The three high-level requirements are considered as fully supported if all subordinate requirements supported. The last row indicates that the aggregated prototypical implementation fully supports the requirements.

	R1	R1.1	R1.2	R1.3	R1.4	R2	R2.1	R2.2	R2.3	R3	R3.1	R3.2	R3.3	R3.4	R3.5
Single Sign-On and Multi-tenancy	◐	○	○	◐	○	○	◐	○	○	○	○	○	○	○	○
Dashboard	○	○	○	◐	○	○	○	○	○	◐	◐	◐	○	○	◐
My-Cases	○	○	○	◐	○	○	○	○	○	◐	◐	◐	○	○	◐
Case Representation	○	○	○	◐	○	○	○	○	○	◐	◐	◐	○	○	◐
Case Summary	◐	○	○	◐	◐	○	○	○	○	◐	○	○	○	●	○
Case Workflow	○	○	○	◐	○	◐	○	◐	○	◐	◐	○	○	○	◐
Flexible Process Adaptation	◐	○	◐	◐	○	◐	○	◐	○	○	○	○	○	○	○
Task Representation	◐	◐	◐	◐	◐	◐	○	◐	○	◐	◐	○	○	○	◐
Custom Data Representation	◐	○	○	○	◐	○	○	○	○	○	○	○	○	○	○
Case Data	◐	◐	○	◐	◐	◐	○	○	●	○	○	○	○	○	○
Case Team	◐	○	○	◐	○	○	○	○	○	◐	○	○	○	○	◐
Case Notifications	○	○	○	◐	○	○	○	○	○	◐	◐	○	○	○	◐
Case Messages	○	○	○	◐	○	○	○	○	○	◐	○	◐	○	○	◐
Case Notes	○	○	○	◐	○	○	○	○	○	◐	○	○	●	○	○
User and Role Management	◐	○	○	◐	○	◐	◐	○	○	○	○	○	○	○	○
$\sum$	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

○ not supported ◐ partly supported ● fully supported

Table 5.1.: Summary of prototypically supported requirements.

CHAPTER 6

Case Studies and Evaluation

The conceptual design and prototypical implementation are evaluated with several integrated care case studies within a Horizon 2020 project. First, the iterative system and model evaluation lifecycle applying the design science framework are presented in Section 6.1. The Personalised Connected Care for Complex Chronic Patients (CONNECARE) project part of the European Horizon 2020 program is introduced in Section 6.2, including accomplished case studies. The conceptual system architecture of CONNECARE is presented in Section 6.3 to demonstrate the applicability of our ACM4IC approach within an integrated environment. Accordingly, the logical system deployment is shown in Section 6.4. The CONNECARE project is a large implementation study which is subdivided into multiple case studies to be performed. The case study modeling process and related artifacts are illustrated in Section 6.5. Similarly, the case study execution behavior is analyzed in Section 6.6. Finally, the results are summarized on a high abstraction level in Section 6.7.

6.1. Iterative System and Model Evaluation Lifecycle

We applied the concept of Plan-Do-Study-Act (PDSA) cycles to ensure a continuously improving prototype. In the planning stage, objectives are declared, the plan is executed in the do stage and changes to it are documented, the result evaluation takes place within the study stage, and adoptions depending on the study results are performed in the act stage. After the first PDSA cycle is completed, the next iteration starts to improve the achieved results interactively. Multiple cycles with slightly different environmental conditions improve the robustness of the prototypes. (Deming, 2018, p. 91)

In our context, we applied the PDSA cycle concept to develop the overall prototype. Additionally, we identified the system- and model-dimensions with a design- and run-time phase each as nested feedback loops that are closely related to the PDSA cycle concept, as illustrated in Figure 6.1. The system run-time phase comprises both model dimensions. Typically, a PDSA cycle lasts six months. The first PDSA cycle started in April 2015 and the last PDSA cycle ended on mid May 2019.

The **System Design-Time Phase** represents an agile software development process and its conceptional steps include design, implement, test, and adjust design or deploy. At the beginning of the prototypical implementation, most changes are triggered from the overall road-map. After three to four overall PDSA cycles, more and more change requests are triggered due to missing modeling capabilities during the run-time phase. Conceptual design changes affecting

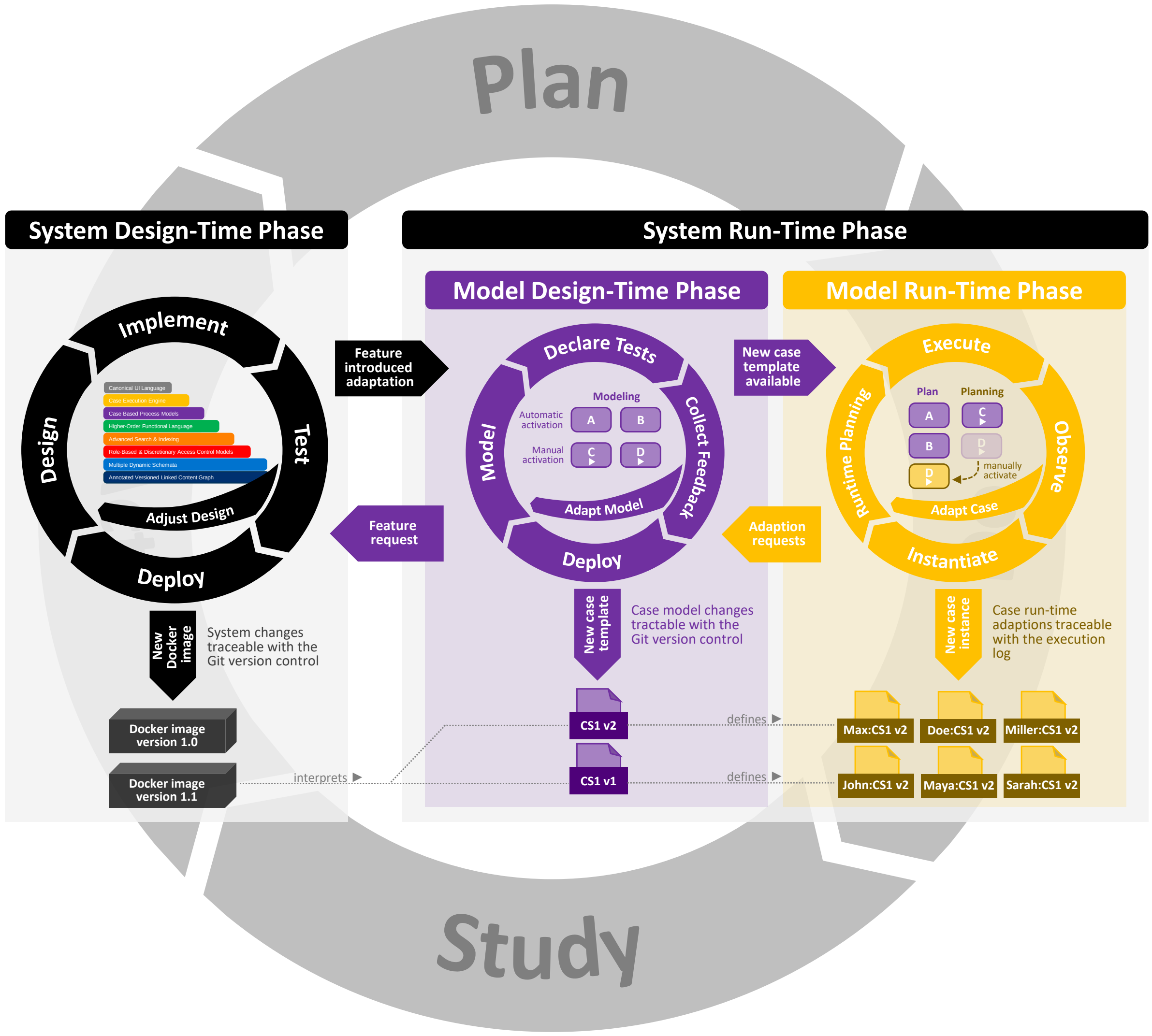


Figure 6.1.: Iterative system and model evaluation lifecycle.

the conceptual layers, including their related meta-model elements are indicated in the center of the cycle. In this model, an enhancement influences the case modeling process, with additional features not being critical, but changed features may lead to massive model migration. All source code changes are tracked with a git repository. As an artifact, deployable versions are built as docker images and distributed on docker hub. Cycle times strongly depend on the applied number and complexity of changes.

The **Model Design-Time Phase** is used to create or adapt existing case templates. All currently deployed system modeling capabilities might be used during this phase. Conceptually, a modeling cycle includes modeling, declaring tests, collecting feedback from end-users, and adapting the model if necessary or the final deployment on a production environment. The modeling includes declaring processes that are automatically activated and typically performed on each case execution and processes that need a manual activation to become executable during model run-time. New case templates typically lead to additional model adaptation requests. Missing modeling capabilities may lead to feature requests on the system design-time phase. The modeling leads to a new case-template artifact that must be deployed to be instantiable. All case templates are declared as XML files and tracked within a git repository. During one system design cycle, multiple model iterations are typically performed.

The **Model Run-Time Phase** allows instantiating case templates. Adaptive Case Management is typically applied on knowledge-intensive use cases where run-time planning is required to handle a case such as patient treatment. Therefore, the initial plan might be adapted during the run-time planning when a case-worker manually activates additional process elements that should be executed, as illustrated in the center of the cycle. After the run-time planning is performed, the case is typically executed. The case-worker continuously observes whether additional adaptations are required and might perform a further run-time planning cycle. Comparing the modeling cycle time, run-time planning is happening ad-hoc and might be performed many times for one case instance. All actions performed on the case instance are traceable within the technical execution log. The resulting artifacts are instantiated cases that represent an individual case, such as patient treatment. Each case instance is defined by exactly one case template and one case template typically has multiple associated case instances. General lessons learned from run-time planning might be incorporated into the next case template version.

To conclude, the design and modeling process contains several iterations on multiple abstraction levels while the cycle times become very short on lower abstraction levels. The cycles was meant as a conceptual instrument while the implementation on lower levels was mostly agile. The evaluation chapter highlights the results on each of the three presented lifecycles in the context of an international integrated care project. In Section 6.3, the conceptual architecture and in Section 6.4, the related system deployment are presented as system design results. In Section 6.5, the case template modeling process and artifacts are presented. Similarly, Section 6.6 focuses on the case execution behavior.

6.2. CONNECARE Project Introduction

Personalised Connected Care for Complex Chronic Patients (CONNECARE)[1] is a European research project that has received approximately five million Euro funding from the European Union's Horizon 2020 research and innovation program under the grant agreement No. 689802. The project started in April 2015 and ends after 45 month in December 2019.

The consortium consists of nine international partners each having either a clinical or a technical background, as illustrated in Figure 6.2. EURECAT, a technology center of Catalonia, coordinates the overall project. The clinical partners are the University Medical Center Groningen (UMCG) located in Groningen, the Assuta Medical Centers (ASSUTA) located in Tel Aviv, the Institut de Recerca Biomèdica de Lleida Fundació Dr Pifarré (IRBLLEIDA) located in Lleida, and the Consorci Institut D'Investigacions Biomediques August Pi i Sunyer (IDIBAPS) located in Barcelona perform implementation studies within their hospitals. The technical partners are the Eurecat Technology Center (EURECAT) located in Barcelona, the Technical University of Munich (TUM) located in Munich, the Advanced Digital Innovation UK Ltd (ADI) located in West Yorkshire, the Università degli Studi di Modena e Reggio Emilia (UNIMORE) located in Modena, and the eWave (EWAVE) located in Tel Aviv provide information technology to perform the implementation studies.

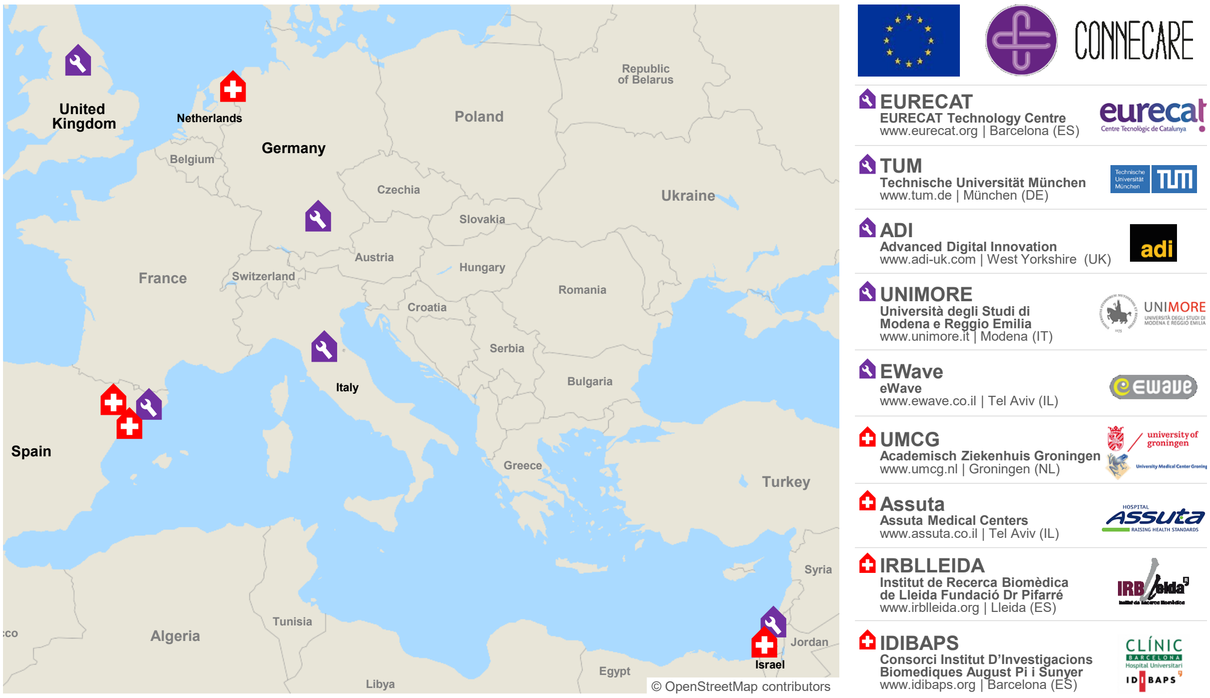

United Kingdom
Netherlands
Germany
Belgium
Poland
Republic of Belarus
Czechia
Slovakia
Ukraine
Austria
Hungary
France
Switzerland
Romania
Croatia
Serbia
Italy
Bulgaria
Spain
Greece
Turkey
Syria
Algeria
Tunisia
Jordan
Israel
Egypt
Libya
© OpenStreetMap contributors
CONNECARE
EURECAT
EURECAT Technology Centre
www.eurecat.org | Barcelona (ES)
TUM
Technische Universität München
www.tum.de | München (DE)
ADI
Advanced Digital Innovation
www.adi-uk.com | West Yorkshire (UK)
UNIMORE
Università degli Studi di Modena e Reggio Emilia
www.unimore.it | Modena (IT)
EWave
eWave
www.ewave.co.il | Tel Aviv (IL)
UMCG
Academisch Ziekenhuis Groningen
www.umcg.nl | Groningen (NL)
Assuta
Assuta Medical Centers
www.assuta.co.il | Tel Aviv (IL)
IRBLLEIDA
Institut de Recerca Biomèdica de Lleida Fundació Dr Pifarré
www.irblleida.org | Lleida (ES)
IDIBAPS
Consorci Institut D'Investigacions Biomediques August Pi i Sunyer
www.idibaps.org | Barcelona (ES)

Figure 6.2.: Geographical distribution of the project consortium, highlighting technical partners in purple and clinical partners in red.

[1] Additional information is publicly available on the project website, http://www.connecare.eu, last accessed in May 2019, and the project summary of the European Commission accessible at https://cordis.europa.eu/project/rcn/202638/factsheet/de, last accessed in May 2019.

The project with an integrated care strategy aims to "[...] co-design, develop, deploy and evaluate a novel integrated care services model for complex chronic patients with smart adaptive case management and self-management technological support" (CONNECARE Consortium, 2019b, p. 3). Target users are chronic patients "[...] with at least one chronic disease, with comorbidities, frail (due to social, economic and/or clinical factors), usually elderly, and who consumes a very high level of health resources" (CONNECARE Consortium, 2019b, p. 3). A more comprehensive project summary is provided by Vargiu et al. (2017) and the transition from connected care to integrated care is described by Kaye et al. (2017).

Conceptually, the project vision is to provide an integrated Smart Adaptive Case Management (SACM) to orchestrate involved stakeholders across organizational boundaries, as illustrated in Figure 6.3. Primary stakeholders are divided into patients, informal carers including patients' relatives, and professionals who are responsible for the patient-centered treatment. According to associated organizations, professionals are further classified into hospital staff, specialist doctors, primary care doctors, and social workers. The patient-centered treatment must be coordinated across all professionals and communicated to the patient. An according, a protocol for the regional implementation of the collaborative management of complex chronic patients is described by Cano et al. (2017). To implements the protocol, professionals are supported with different integration concepts, such as telemonitoring, self-management, and decision support. Telemonitoring enables the detection of anomalies in an early stage and enables immediate reactions. Patient empowerment is ensured with an integrated Self-Management System (SMS) (Vargiu et al., 2018b,a, 2019a). Furthermore, the decision support (Mariani et al., 2019) provides professionals with contextual information to rationalize decisions, and a recommender provides recommendations for patient self-management (Fernández et al., 2017). The presented vision leads to a three-dimensional paradigmatic shift, which includes an organizational, a case services, and a technological shift.

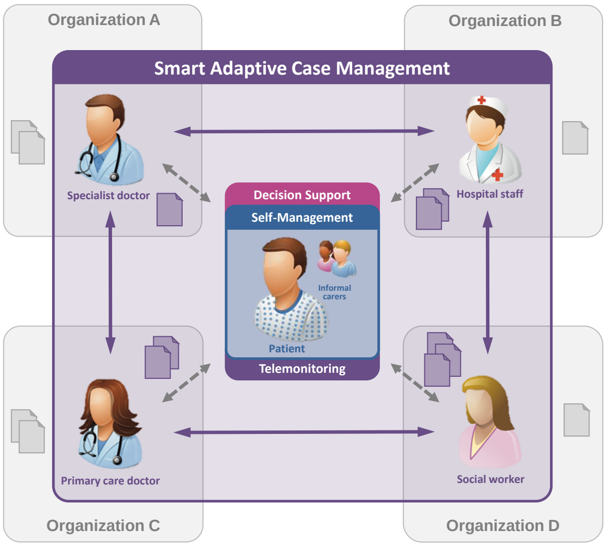


Figure 6.3.: High-level project vision adapted from CONNECARE Consortium (2016). The Smart Adaptive Case Management is composed of the collaborative, purely meta-model-based ACM4IC approach enriched with clinical decision support.

The project "[...] is being implemented and studied in the context of real life deployment in 4 real sites (Barcelona, Lleida, Groningen, Tel Aviv) going beyond controlled pilots is an important added value and differentiation from other ongoing integrated care projects" (CONNECARE Consortium, 2019b, p. 18). As an ACM solution, the collaborative, purely meta-model-based ACM4IC approach presented in this thesis is applied, except in Barcelona where a self-implemented solution is used. Project achievements are evaluated with implementation studies. Primarily, two to three case studies are accomplished on each site. The heterogeneity of the project "[...] implementation sites regarding processes, integration, logistics, etc. is real world integrated care it is a big challenge both from the clinical (co-design, studies, evaluation) and the technological (implementation, customization, integration) side, it truly prepares this consortium for large scale deployment and transferability/replicability of model and solution" (CONNECARE Consortium, 2019b, p. 18).

Formally, the project is decomposed into nine Work Packages (WP), as illustrated in Figure 6.4. Each WP has a responsible WP leader to coordinate the work between the WP participants, indicated within squared brackets. The dark gray WPs primarily focus on content, whereas the light gray WPs mainly focus on management aspects. Arrows indicate main the dependencies between the WPs. The objective of each WP is briefly described in the following. WP2 focuses on the co-design of the integrated care services considering the clinical environment in close collaboration with WP3, the Smart Adaptive Case Management for professionals, and WP4 focuses on the Self-Management System for patients. WP5 focuses on the integration with hospital information systems. WP6 prepares interdisciplinary implementation studies. WP7 targets the implementation study evaluation and upscaling. Central coordination efforts are bundled within WP1. In WP8, project results are disseminated and communicated. Moreover, new business models are explored. Ethical aspects strongly influence the project considered in WP9.

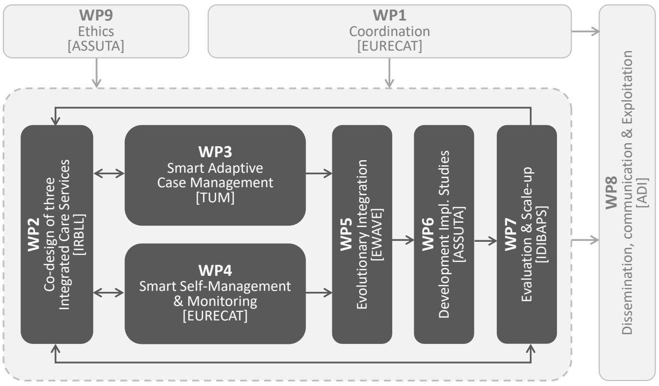


Figure 6.4.: Workpackages adapted from CONNECARE Consortium (2019b).

Related integrated care projects are NEXES (Cano et al., 2015), ICT4LIFE (Osvath et al., 2017), PolyCare (Velasco et al., 2016), ProACT (Doyle et al., 2017), and CAREGIVERSPRO-MMD (Tzallas et al., 2018). Similarities and differences of the ICT4LIFE and CAREGIVERS-PRO-MMD projects across different dimensions, such as project aims, users monitoring, and the underlying architecture are provided by Solachidis et al. (2018). However, the CONNECARE project has a stronger focus on an integrated, collaborative, and flexible adaptive case management approach.

6.2.1. Case Studies in Groningen

In Groningen, "[...] the implementation of the organizational model [...] consist of the case management model to support citizen/patient empowerment and enable clinicians to continue to support care beyond the hospital walls (i.e. after discharge). Although using the same technology (SACM&SMS) developed in the CONNECARE project, CS1 and CS2 have a different focus regarding self-management and monitoring." (CONNECARE Consortium, 2019c, p. 29f.)

Groningen CS1
Within the CS, "[...] the SMS will be primarily used to promote and support the self-management abilities of community dwelling asthma and COPD patients. To this end, the most important elements of the CONNECARE systems that have to be developed are sharing of medical records and results of diagnostic tests, digital questionnaires sent via the SACM and visualised in the SMS, a chat function to communicate with care providers and links to reliable website for information on disease, treatment and prevention. In addition, monitoring of physical activity is of importance, for which we will be using the Fitbit." (CONNECARE Consortium, 2019c, p. 29)

Groningen CS2
Within the CS, "[...] the organisational model focuses primarily on monitoring vital signs of oncological patients through mobile devices. As such, the most relevant items such as physical activity, temperature and weight have been defined to be included in the first version of the SACM and SMS in order to start including patients in the implementation study." (CONNECARE Consortium, 2019c, p. 29)

6.2.2. Case Studies in Tel Aviv

In Tel Aviv, the "[...] organizational model for integrated care [...] is essentially a case management model focused on integration and continuity of care between the hospital, primary and secondary healthcare in the community and social care. Due to the structure of the Israeli healthcare system this means not only collaboration of professionals and integration between levels of care but also collaboration between different organizations – in the case of CONNECARE, Assuta Ashdod Hospital and Maccabi Healthcare Services and the Ashdod municipality department of social services." (CONNECARE Consortium, 2019c, p. 25)

Tel Aviv CS1/CS2
Considering "[...] the challenges of integrating the SACM with very complex existing legacy systems in both Assuta and Maccabi, it was decided that the SACM would be used exclusively by the Assuta nurse case manager and the physical therapy staff for CS2 (patients scheduled for elective surgery) and by the Maccabi nurse case managers for CS1 (patients admitted via the Emergency Department). The coordination and integration with the other professionals caring for the patients is done by the nurse case managers using the Assuta and Maccabi Electronic Medical records systems respectively, and by direct communication via phone or emails. The SACM is used by the case managers for initial and ongoing eval-

uation of the patients using agreed upon measurement tools and documenting the care plan, prescribing patient tasks that are transmitted to the patient's SMS and monitoring the data fed back from the SMS. The SACM is also used for communication between the three case managers and the physiotherapists, and for the purpose of documenting the actions taken by the staff regarding the patient (conversations with the patient or his family, communication with other care givers, etc.)." (CONNECARE Consortium, 2019c, p. 26)

6.2.3. Case Studies in Lleida

In Lleida, the "[...] organizational model for integrated care [...] consists of a case management model focused on integration and continuity of care between the hospital and its network of primary care centres as well as social services. This requires the collaboration of professionals from different centres and organizations: Hospital Universitari Arnau de Vilanova (Lleida), Hospital Universitari de Santa Maria (Lleida), and a network of over 20 primary care centres covering the city of Lleida but also the whole health region of Lleida (≈ 400k people)." (CONNECARE Consortium, 2019c, p. 28)

Lleida CS1
Within the CS, "[...] patients are recruited in the emergency room of the Arnau de Vilanova Hospital / Santa María Hospital or during the subsequent hospital admission but always before discharge from the hospital. The hospital's case manager is responsible for the recruitment of patients. The hospital discharge of patients in the CONNECARE program is planned in agreement with primary care personnel of the centre corresponding to each patient. Therefore, there is a crossfertilization between the day-to-day knowledge on the patient of the primary care team, and the more specialized hospital team managing the specific hospital admission. Patients benefit from the SMS CONNECARE application for smartphone or tablet during the 90 days after discharge. Throughout the process, the management of CONNECARE patients by hospital and primary care staff is carried out through the SACM [...]". (CONNECARE Consortium, 2019c, p. 28)

Lleida CS2
Within the CS, "[...] the hospital's surgery team together with the hospital case manager are responsible of identifying and recruiting the eligible patients. When considered necessary, the surgery team designs a pre-habilitation plan to be monitored by primary care. After surgery and during hospitalization, patients in the CONNECARE program receive the SMS CONNECARE application for smartphone or tablet, a rehabilitation and physical evaluation plan and a pain control plan. Throughout the process, the management of CONNECARE patients by hospital and primary staff is carried out through the SACM system developed by CONNECARE. All patients undergo an exhaustive follow-up for 1 month coordinated by the hospital, after which an additional 2-months follow-up is coordinated by the corresponding primary care team begins." (CONNECARE Consortium, 2019c, p. 28)

6.2.4. Case Studies Related Stakeholders

An integrated care environment consists of multiple organizations with several involved stakeholders from different disciplines and their coordination is challenging. Within an onion model according to Alexander (2005), the CONNECARE consortium identified three stakeholder environment abstraction layers. In the core, all stakeholder directly using the CONNECARE system are placed, i.e., medical professionals and the treated patient using the system. The surrounding system includes all stakeholders directly affected by the CONNECARE system. I.e., an ethics commission who must confirm that the system functionally complies with the concerning ethical and legal conditions. The wider environment contains all stakeholders who might indirectly be affected by the CONNECARE system. I.e., hospital providers who would like to collaborate and provide or use interfaces for information exchange to extend the integrated care solution. Figure 6.5 shows the environmental stakeholders identified in Lleida. The onion core with the stakeholders directly using the CONNECARE system is colored in purple. Identified and engaged stakeholders are highlighted whereas identified, but not yet engaged stakeholders are faded. Healthcare professionals using the SACM system are decorated with a plus icon, while patient stakeholders using the SMS system are decorated with a heart icon. For selected stakeholders, a more detailed description regarding their individual value is presented in Table A.1. For the case studies in Groningen and Tel Aviv, similar onion models with slightly different stakeholders exist. However, due to the organizational setting, fewer care professionals are actively engaged there (cf. Section 6.6.5, which analyzes the actual case team members and roles across all case studies during case execution).

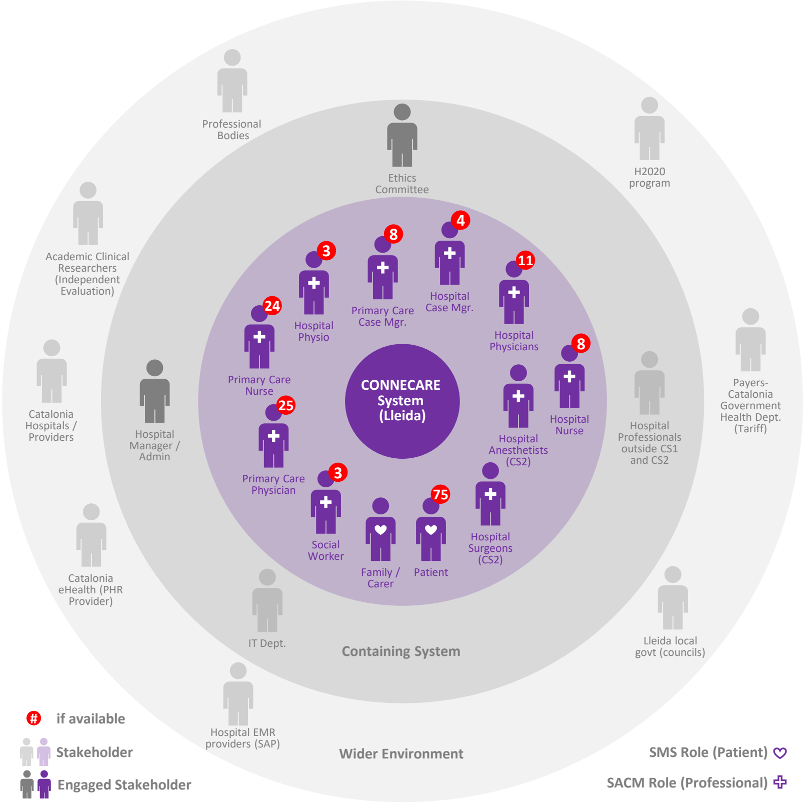


Figure 6.5.: Stakeholders in Lleida, adapted from CONNECARE Consortium (2019b) and enhanced with numbers where available from CONNECARE Consortium (2019a).

6.3. CONNECARE System Architecture

This section describes the high-level architecture of the CONNECARE Project. Figure 6.6 illustrates the architecture with components, neighbor systems, and relevant roles. The CONNECARE system is mainly composed of the Smart Adaptive Case Management (SACM) system used by medical professionals and the Self Management System (SMS) used by patients. Care professionals use the SACM as a patient-centric collaborative case management platform that allows direct and flexible interaction with the patient via the integrated SMS system.

The SACM is composed of multiple components that are primarily a web-based professional end-user interface, the ACM4IC engine that provides the core backend functionality, and a clinical decision support system that provides additional risk information for a patient. The professional end-user interface is implemented with Angular[2] version 5.2.10 as a Single Page Application (SPA). The ACM4IC engine is developed using Java[3] and provides modeling capabilities on different conceptual layers, as indicated with colors in Figure 6.6. The functionality of those layers is exposed on a JSON-based[4] API (cf. Section 5.5.1). The SACM architecture follows a purely meta-model-based approach to enable reusing single components in different contexts, which implies the frontend as well as the integration strategy.

The SMS is composed of several micro-services that represent the SMS backend and an Android and iOS App for the patient. The apps are created with the Xamarin[5] which allows using a single code base for Android and iOS. Each functionality is represented within one micro-service, such as the patient monitoring prescriptions. On the SACM, a `DualTask` meta-model element is used to interact with this micro-service. The integration pattern of declaring models on the SACM to connect hard-wired micro-services on the SMS is repeatedly used and provides the required degree of flexibility.

The centralized user identity management (UIM) interface is integrated into the SACM professional end-user interface as one view providing seamless integration for the end-users. All CONNECARE services use the JSON Web Tokens (JWT)[6] generated by the UIM which simplifies the integration across services on all architectural levels. Therefore, the professional user interface can access the UIM API endpoint for the integration directly.

All components use SQL-compatible[7] databases to persist information. This allows sharing the infrastructure for persisted information and simplifies the maintenance and backup strategy. The communication between the SACM and the SMS is realized with a message broker named RabbitMQ[8] that provides a producer and consumer API endpoint.

Integration with third-party site-specific hospital information systems is accomplished with adapters that push the needed information to the message broker. Due to legal regulations, only read operations are performed on hospital information systems.

[2] https://angular.io, accessed on June 5, 2019
[3] https://www.java.com, accessed on June 5, 2019
[4] https://tools.ietf.org/html/rfc8259, accessed on June 5, 2019
[5] https://dotnet.microsoft.com/apps/xamarin, accessed on June 5, 2019
[6] https://tools.ietf.org/html/rfc7519, accessed on June 5, 2019
[7] https://aws.amazon.com/rds/aurora, accessed on June 5, 2019
[8] https://www.rabbitmq.com, accessed on June 5, 2019

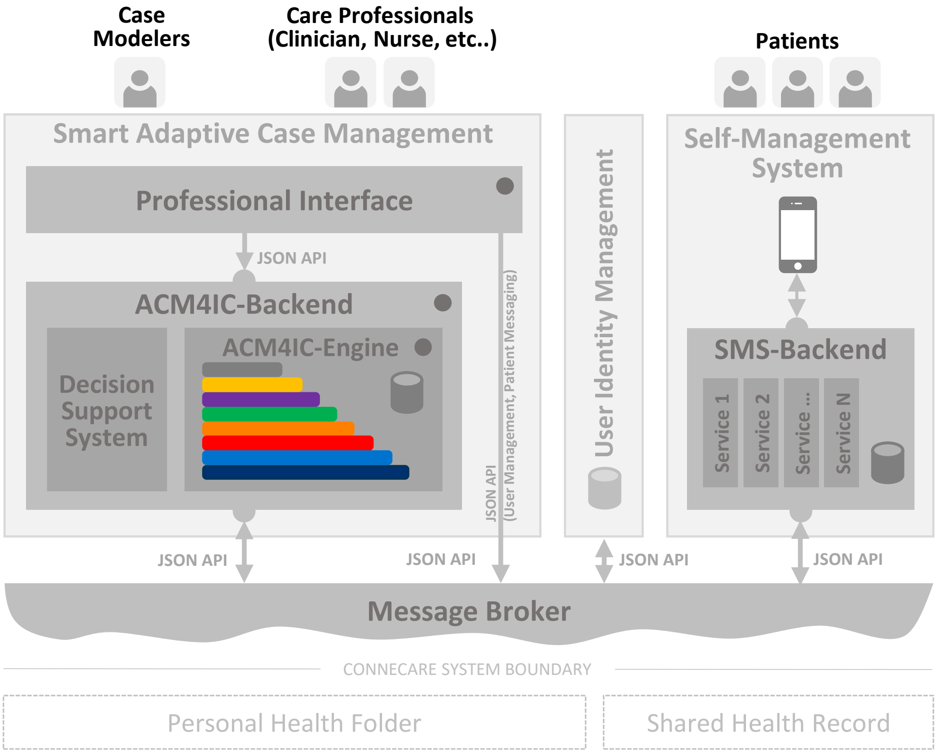


Figure 6.6.: CONNECARE System architecture adapted from Michel and Matthes (2018). Generic ACM4IC components are highlighted with a dot in the upper right corner.

Table 6.1 declares the primary system responsibilities across the CONNECARE system. Each column represents a conceptual CONNECARE component, such as the UIM, the SACM, and the SMS, while each row describes a responsibility. The responsibilities are assigned to the conceptual system components either as a master, which means the system is primarily responsible or as a slave, which means another system is primarily responsible and data may be synchronized.

The UIM primarily manages the CONNECARE Single Sign-On (SSO) for all users. After successful authentication, a JWT token is generated with a private key. All other components can only validate that JWT token with a public key. The UIM similarly leads creating a new user, updating an existing user or disabling an existing user. When a new user is created, the UIM creates the user internally and pushes the newly created user to the SACM system. All other user operations are performed similarly.

The SACM primarily manages all case-related information. The case authorization determines if a user is allowed to perform case read operations such as accessing a task, case write operations such as completing a task, or heavy case owner operations such as deleting a case. The SMS requests the case authorization information if needed on the fly to provide consistency and to avoid unnecessary synchronization. Operations such as the case instantiation, case updates, or case delete operations are primarily managed by the SACM and synchronized with the SMS. During modeling time, hooks for several events might be defined that enable synchronizing the SMS. A typical pattern is a hook on completion of a monitoring prescription task which needs to be performed on the SMS, or a hook on completion of the case.

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

Table 6.1.: High-level system responsibilities.

Figure 6.7 illustrates the conceptual orchestration of a monitoring prescription task across the SACM and SMS systems. A monitoring prescription task is used to monitor either the patient's systolic and diastolic blood pressure, heart-beat, weight, or body temperature. The related `DualTask` is modeled in the SACM and interacts with the SMS monitoring prescription microservice to provide seamless integration. A `DualTask` is a combination of a `HumanTask` performed by a care professional followed by an `AutomatedTask` performed by the patient. The integration from SACM to SMS uses hooks defined in the model that perform HTTP requests on process state transitions. The SMS to SACM integration uses traditional API endpoints to update the needed resources.

In the following, the conceptual flow of a monitoring prescription is elaborated. Imagine the cases meet all preconditions to perform a monitoring prescription task of the type `DualTask`. The nurse Anne Williams manually adds a task to the workplan stage that semantically changes the state of an enabled monitoring prescription to active. Anne Williams selects the measurement type *body temperature*, defines the duration with a start and an end date, selects the measurement frequency *once a day*, types a minimum and maximum temperature which should not be exceeded, and completes the human part of the monitoring prescription `DualTask`. After the SACM successfully validated and persisted all input parameters, the SACM checks if the state transition triggers a hook execution. The state transition on completion of human part triggers the HTTP hook execution with a URL that refers to an SMS API endpoint. The persisted task is attached as JSON serialized payload. Every new monitoring prescription on the SMS is pushed to the SMS mobile application.

Next, the patient Maya Wilson uses her wearable thermometer and measures her body temperature. The resulting data is automatically transferred to the SMS backed, which drafts the monitoring prescription task in the SACM backend. Depending on the defined measurement duration and frequency, the patient needs to repeat that step many times over a long time period and the nurse can access the results in the SACM UI almost in real-time.

Depending on the patient's measurement values and the nurse's defined thresholds, the measurements may exceed the allowed thresholds and the SMS creates a custom notification on the SACM to inform the related task owner. When the nurse Anne Williams accesses her dashboard, a new notification is shown for that incident. Clicking on the dashboard notification directly opens the monitoring prescription details task page where the measured data is graphically represented with a line chart. After accessing the measured values, Anne Williams notices that the thresholds for the chronic patient Maya Wilson are too low and uses the task correct feature to increase the value. The SACM backend receives the corrected information and may send an HTTP hook to the SMS if defined.

When the monitoring prescription is expired, the SMS completes the automated task part with the latest measurement values. Actual case executions may contain several more exceptional interactions that could contain direct patient messaging to understand what caused the out of range measurement values, the case team messaging to discuss the acceptable thresholds, the case notes to document gained knowledge, and possibly the creation of an additional monitoring task to track the blood pressure in parallel.

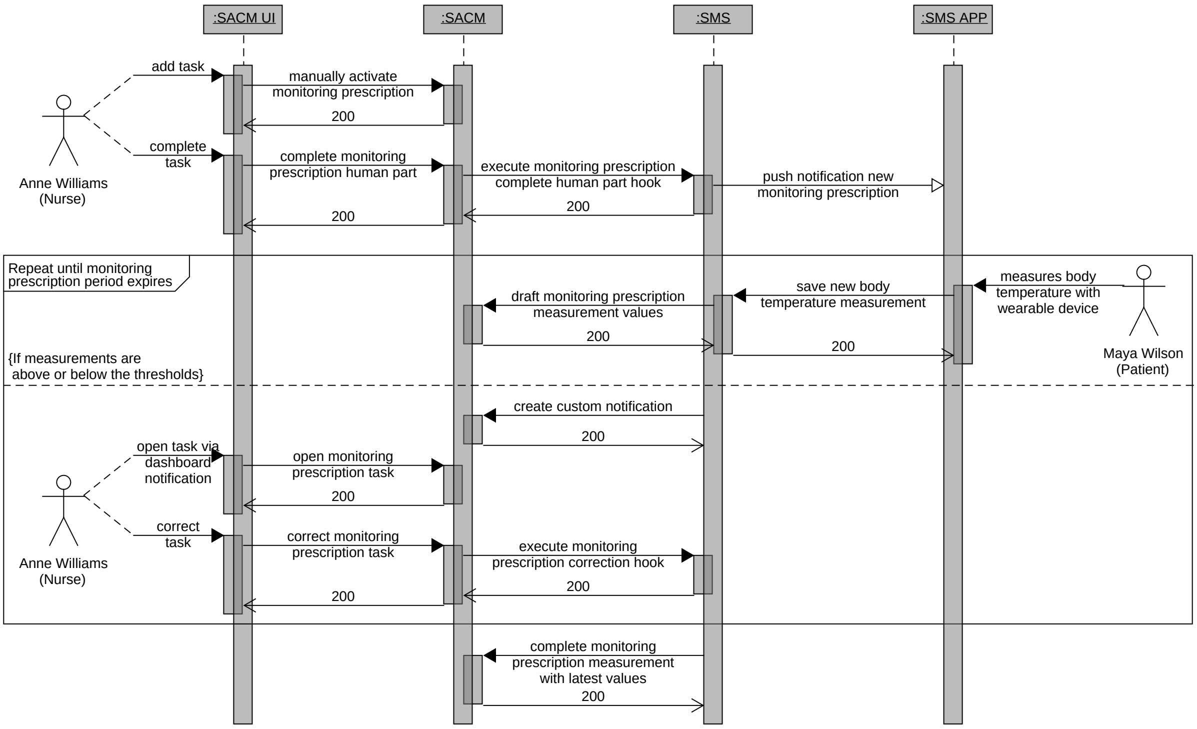

:SACM UI
:SACM
:SMS
:SMS APP
add task
manually activate monitoring prescription
200
Anne Williams (Nurse)
complete task
complete monitoring prescription human part
execute monitoring prescription complete human part hook
push notification new monitoring prescription
200
200
Repeat until monitoring prescription period expires
measures body temperature with wearable device
save new body temperature measurement
draft monitoring prescription measurement values
200
200
Maya Wilson (Patient)
{If measurements are above or below the thresholds}
create custom notification
200
open task via dashboard notification
open monitoring prescription task
200
Anne Williams (Nurse)
correct task
correct monitoring prescription task
execute monitoring prescription correction hook
200
200
complete monitoring prescription measurement with latest values
200

Figure 6.7.: Conceptual orchestration of a monitoring prescription task across systems.

6.4. CONNECARE System Deployment

The CONNECARE project consortium consists of nine Partners. Five are technical partners that are geographically distributed across multiple countries. Additionally, two technical partners apply nearshore outsourcing to support development activities. The organizational coordination effort delivering an integrated care software platform is high. Considering that the CONNECARE system partly uses extends existing services of the partners, it is not reasonable to restrict technology to a particular stack. Collaboratively, the technical partners early decided to virtualize the deployment with docker[9] containers to simplify the integration and deployment processes. The docker virtualization concept enables encapsulating services on a host system and sharing host resources where required. Furthermore, services can be tested locally within an exactly similar environment before the actual integration is performed, which increases the robustness. Additionally, dockerization in combination with Kubernetes[10] would allow the orchestration of a dynamically scalable system.

Figure 6.8 illustrates the logical CONNECARE system deployment on the Amazon cloud. Each dockerized service is illustrated with a container on the logical AWS EC2[11] component. Infrastructure containers are illustrated in gray and actual services are highlighted in purple. The purple services can be categorized into three conceptual blocks the SACM, the UIM, and the SMS which are identified with container name prefixes. The SMS micro-service architecture approach leads to a large number of services, while the SACM provides fewer services due to a high cohesion between logical modules. The services groups can be managed with different docker compose files. All services use the AWS Aurora[12] database service that provide an SQL-compatible interface to simplify the database management and backup strategy. File attachments are stored within the EC2 instance. The integration with the local hospital information systems is applied via publicly available APIs of the CONNECARE System.

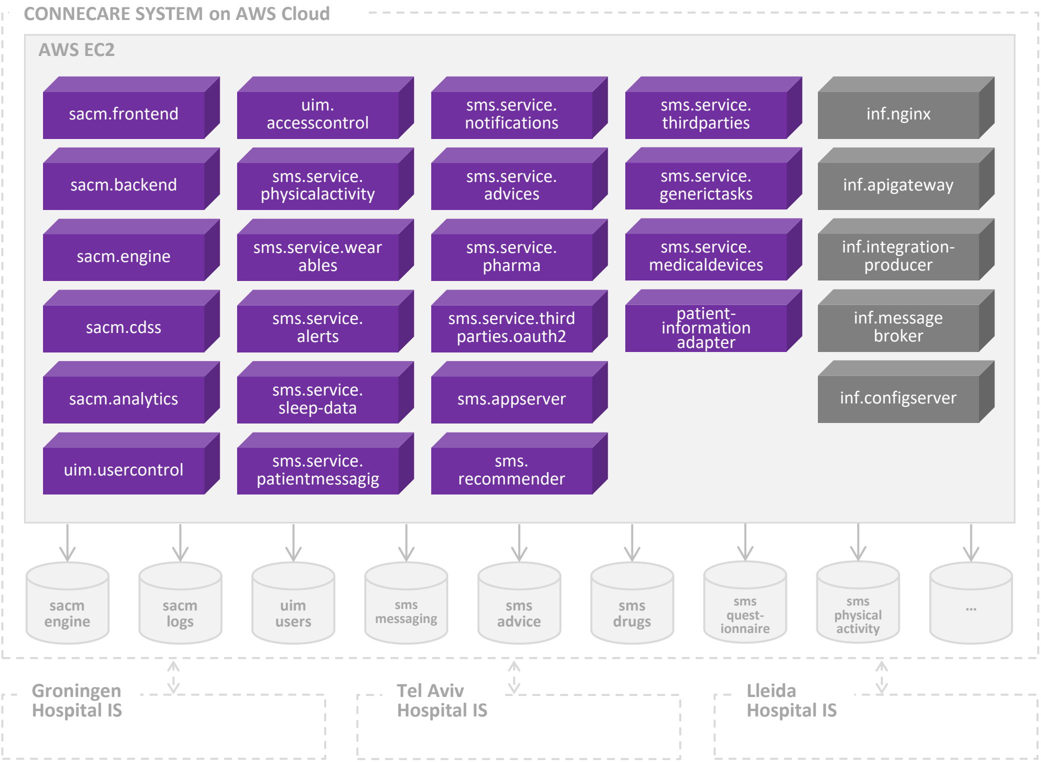


Figure 6.8.: Dockerized deployment on AWS.

[9] https://www.docker.com, accessed on June 5, 2019
[10] https://kubernetes.io, accessed on June 5, 2019
[11] https://aws.amazon.com/ec2, accessed on June 5, 2019
[12] https://aws.amazon.com/rds/aurora, accessed on June 5, 2019

Figure 6.9 illustrates the SACM deployment process. All code artifacts are managed with git repositories either on GitHub[13] for the publicly available repositories or on BitBucket[14] for the private repositories. Each partner manages their repositories to have full control. At the beginning of the project, the SACM build process was fully automated to provide a complete continuous integration pipe but this leads to too many restarts of the dockerized services on the test environment which may then interrupt a long-running case definition import script. Today, the build process is semiautomated to keep the effort low while having the maximum flexibility. After the local development is pushed to the related repository, the developer triggers the build process script on the build server via ssh console. As optional input, the docker build tag can be passed while as default, the docker tag *latest* is used. First, the latest changes in the git repository are pulled and then the actual build process is started. During the build process of the sacm.engine[15] docker image, version information that contains the date, the git commit hash, and the docker tag of the build is generated. This build information is accessible on the deployed docker image via REST API and enables to exactly map the built version to a git commit. This information is mostly useful for bug reporting or for detecting incompatible model versions. After the script completed the actual build process, the related docker image is created and pushed to docker hub[16]. In the last step, the new docker image is deployed on either the development instance, the test instance or on the production instance. The development environment is hosted on a EURECAT internal server, whereas the test and production environments are hosted on an Amazon EC2 server. The test environment automatically pulls the SACM docker images with the tag latest every view minutes and restarts the docker containers if necessary. On the production environment, the docker tags contain the release version and the release date. This implies that the docker compose files must be adapted before each deployment. In the case of occurring compilations, the previous docker images can be used again easily if no data migration was necessary. Within the whole CONNECARE project, the build processes are very heterogeneous due to the organizational settings, but all deployment processes use docker hub to store the final docker image.

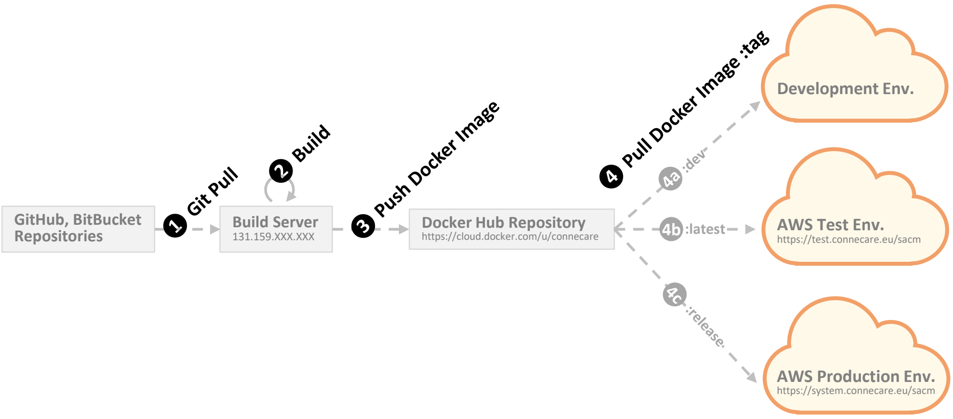


Figure 6.9.: Conceptual SACM build process.

[13] https://github.com, accessed on June 5, 2019

[14] https://bitbucket.org, accessed on June 5, 2019

[15] formerly named sacm.sociocortex

[16] https://hub.docker.com/u/connecare, accessed on June 5, 2019

6.5. CONNECARE Implementation Studies Case Modeling

The case studies depend on case templates modeled for the specific case study needs. The resulting case template structure is notated as extend CMMN and elaborated in Section 6.5.1. In addition, the models used for healthcare service integration and orchestration are presented in Section 6.5.2 in more detailed. Modeling is a complex endeavor and is accomplished in multiple iterations, as shown in Section 6.5.3.

6.5.1. Declared Case Templates

Conceptually, we modeled the case templates according to the protocol for collaborative management of complex chronic patients published by Cano et al. (2017). The identified abstract and mainly sequential workflow steps are illustrated in Figure 6.10 and additionally, they are colored according to the conceptual degree of structure (cf. Figure 3.1). The structured case identification as the first step identifies potential case study patients fulfilling the inclusion criteria. In the second step, the case evaluation, patient information required to define a holistic action plan is primarily gathered. Furthermore, an adherence profile and requirements for the self-management are collected. Additionally, socio-demographic barriers and logistic barriers are documented. Conceptually, the case evaluation is structured and allows ad-hoc exceptions to react to patient-specific needs. Depending on the information gathered, an individual patient-centric workplan is defined in the third step, the case evaluation. In the following fourth step, the patient executes the workplan, the professional follows up, reacts on occurring events, and dynamically adapts the personalized workplan where needed. The workplan execution step is unstructured and assembled based on predefined fragments within the personalized workplan definition. In the last step, the structured patient discharge is performed.

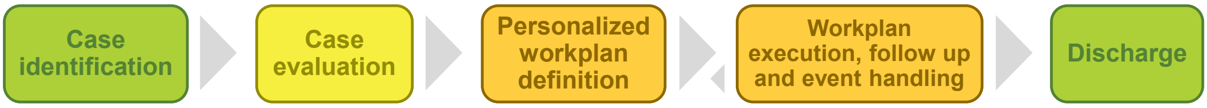

Case identification
Case evaluation
Personalized workplan definition
Workplan execution, follow up and event handling
Discharge

Figure 6.10.: Protocol for collaborative management of complex chronic patients adapted from Cano et al. (2017) and colored according to the degree of structure in Figure 3.1.

However, site and case study specific requirements significantly influence the practical case template modeling. We depict the latest case template versions of Tel Aviv CS2 and Lleida CS2, represented in Figure 6.11 and Figure 6.12 respectively, to highlight diversity within the declared case templates. Additionally, all other case study templates are notated and visually illustrated in Section A.3.

Immediately after a `Case` is instantiated, the case identification stage without `SentryDefinitions` is active. First, the patient is assigned and the predefined professional case roles are selected. The resulting roles are used to predefine task ownership within the case. A `HttpHookDefinition` listens on the `HumanTask` completion to synchronize the new information with the SMS. To reduce the danger of confusion, multiple `SentryDefinitions` prevent that professionals proceed to add more patient information without an assigned patient. All case studies require that patients use the SMS mobile application on a smartphone. Therefore a technological test measures the patients' ability to do so. Besides the required patient consent, additional site-specific tasks are declared. Completing the case identification stage satisfies multiple `SentryDefinitions` and enables or activates pending stages.

In Tel Aviv CS2, the case identification state has multiple cascaded `SentryDefinitions` enforcing a particular task execution order. After completing the case identification stage, the repeatable case evaluation stage is automatically activated for the first iteration. The declared serial repetition enables care professionals to reevaluate the patient when required at any time after the first evaluation is accomplished, to understand how the treatment affects patient health. The decision whether and when an evaluation repetition is required must be decided individually by a care professional. Therefore, activation is modeled with an expression automatically activating the first enabled case evaluation iteration, whereas the second iteration needs a manual activation triggered by a knowledge worker. Within the case evaluation stage, multiple required `HumanTasks` are used to evaluate the patient status.

In the latest case template version, the workplan stage is activated directly after the case identification and simultaneously with the first iteration of the case evaluation. It allows several roles working in parallel, defining the first obvious individual workplan tasks before the first evaluation stage is completed. Compared to the conceptual steps described above, the workplan definition and workplan execution are modeled within the workplan stage. The knowledge-intensive run-time planning is performed by clinical professionals who manually activate one or multiple predefined `DualTasks` to create a patient-centered treatment plan. Every `DualTask` which might be a physical activity, monitoring prescription, drug prescription, patient questionnaire, simple task, or advice first requires a human input from a clinical professional to declare boundary conditions and the patient must perform the action described within that task. Technically, hooks are used to push the task information to the SMS system that interacts with the patient via a smartphone application. A detailed description of those modeled `DualTask` types is presented in Section 6.5.2. When the patient's treatment is accomplished, the discharge stage is activated and the discharge form can be completed.

In the following, we focus on the conceptual particularities in Lleida CS2. The case template separates the pre- and post-hospitalization workplan stages. Separating those stages enables to distinguish between needs before and after hospitalization, which enables further customizations. I.e., physical activities are only foreseen in the post-hospitalization phase, whereas for drug prescriptions the opposite applies. Additionally, the workplan contains optional and mandatory `HumanTasks` which must be completed by the care professional. To summarize, the workplan definition in Lleida CS2 provides more guidance for care professionals and offers additional run-time planning options. The case evaluation stage contains several more tasks with specific tasks requiring a manual activation. Furthermore, a case evaluation repetition is not allowed.

Depending on the case study, the case templates differ significantly even if the extended CMMN notion is comparably similar. The similarly named tasks are adjusted to the local needs, which might be an additional derived attribute to calculate a hospital-specific score. The options of task parameters might be adapted by extending or removing enumeration options on the linked attribute. Finely adjusted adaptations are applied to optimize the daily treatment process in practice, which includes the customization of default values according to expected patients. Typically, custom representations are used to adapt default representations to case study specific needs. Additionally, modeled roles heavily depend on the organizational context. Most case templates are using the local language to reduce the usage barriers, whereas the technical model names are provided in English to ensure maintainability.

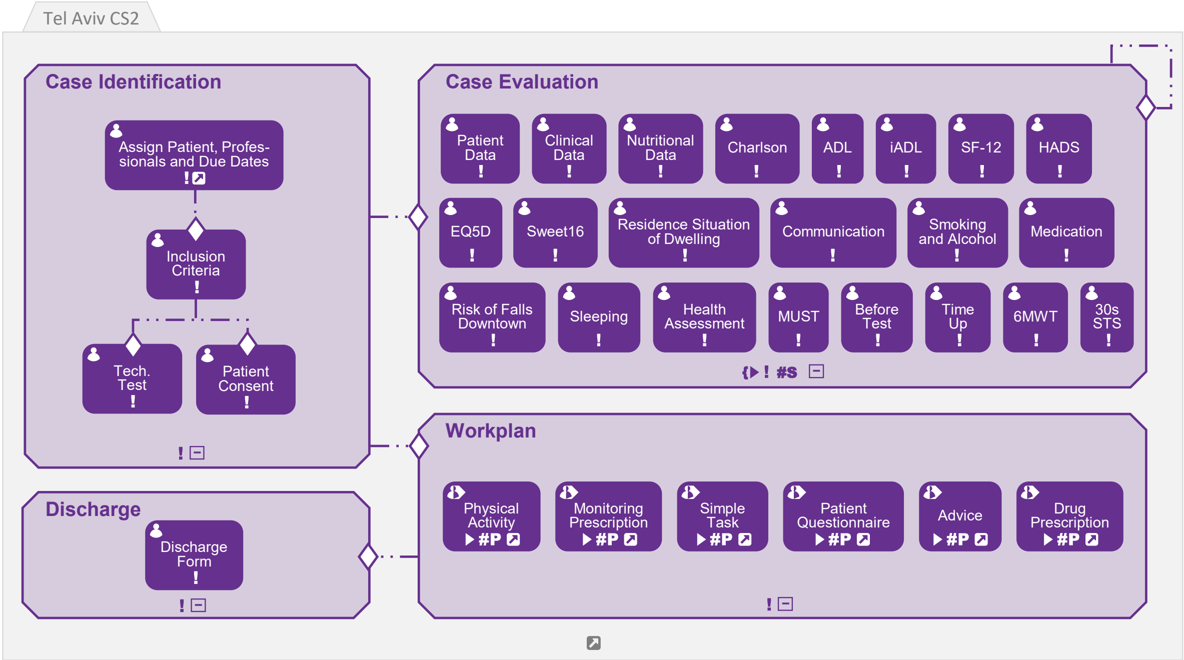


Figure 6.11.: Case template Tel Aviv CS2.

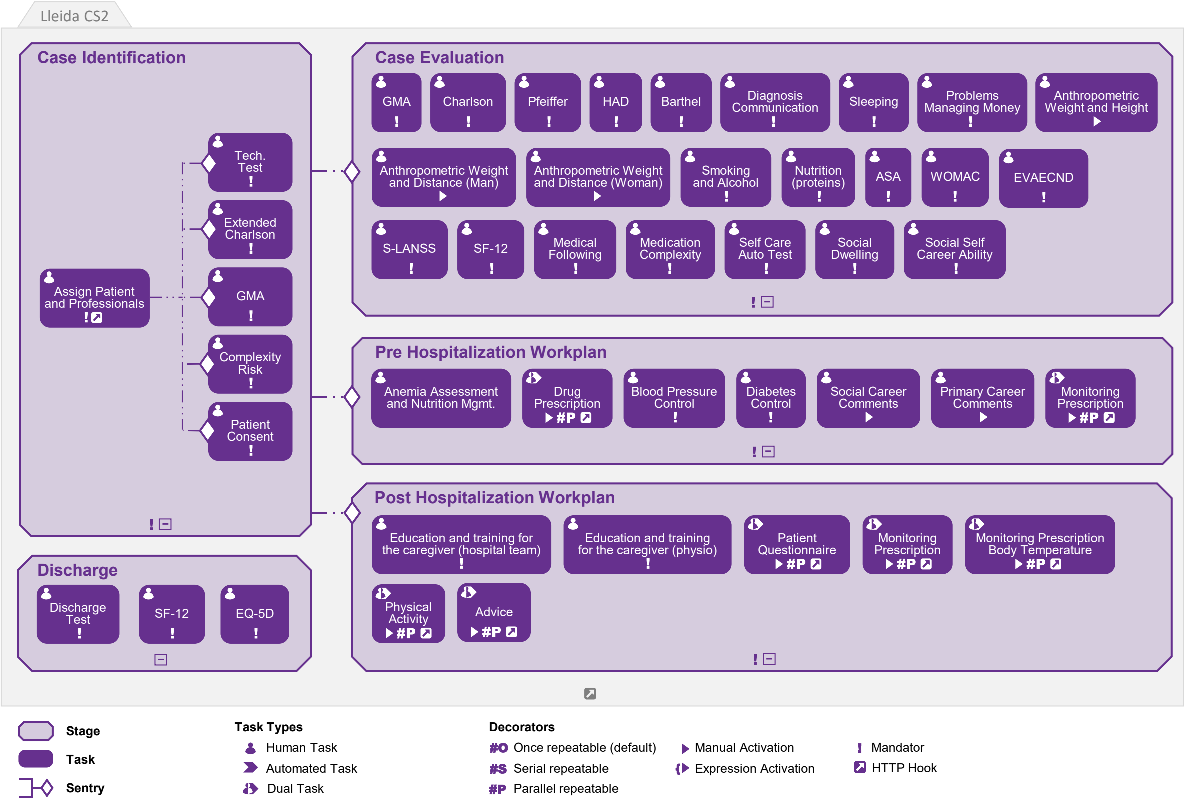


Figure 6.12.: Case template Lleida CS2.

6.5.2. Used Model Elements to Orchestrate Healthcare Services

Multiple micro-services must be integrated seamlessly and orchestrated with the case execution engine to enable a patient centered-treatment. We only used modeling elements to achieve seamless integration and orchestration. The use of modeling methodologies increases the flexibility to customize the integration to site-specific requirements without additional programming effort. The healthcare services orchestration is typically required for individual patient treatment during the workplan stage execution, which is adaptable to patient needs with run-time planning. All healthcare services are orchestrated with the `DualTaskDefinitions` in combination with `HttpHookDefinitions` (cf. Section 6.3 and 6.4). First, the professional declares certain meta information required for a specific micro-service to start, i.e., start date and end date. Second, upon completion the human `DualTask` part sends the information to the micro-service. Third, the micro-service regularly transmits information to the case execution engine until the automated part is accomplished. For reason of usability, all possible micro-service options are grouped within one `DualTask` that allows care professionals to choose the detailed options from an enumeration attribute when performing the task. The `Alert` concept, which is named notifications in the end-user interface, allows that micro-services create domain-specific task-related notifications to inform care professionals about patient anomalies. In the following, the related `DualTasks` modeling is described:

- **Monitoring Prescription** A monitoring prescription allows care professionals to monitor a patient's blood pressure, body temperature, heart rate or weight between an individually declarable start and end date. Furthermore, professionals declare the measurement frequency and the unit, which might be a day, a week, or a month. Depending on when the measurement should be performed, time slots such as breakfast, lunch, afternoon snack, dinner, before sleep, anytime during the day are declared. Additionally, the professionals declare a minimum and maximum threshold value and when one of those is exceeded the related monitoring micro-service creates a custom domain-specific `Alert` to notify the responsible clinician. A maximum delay expressed in minutes to fulfill the prescription before an `Alert` is created might be declared. While the monitoring is active, the care professional receives one or more line chart diagrams representing the measurement results. The resulting measurement values are stored within a JSON typed attribute that contains a `uiReference`, which is conceptually named `CustomDataRepresentation`, allowing the end-user interface to interpret it as a diagram. Considering that a blood pressure measurement includes the systolic and diastolic values, the minimum and maximum thresholds must be declared for both. Therefore, the `uiReference` conditional multiplicity overrides the attribute default multiplicity and allows to show a second minimum and maximum threshold field depending on the currently chosen monitoring prescription type.
- **Drug Prescription** A drug prescription allows care professionals to prescribe and monitor the intake of drugs. A massive number of drugs exists and depending on the region, different drugs are prescribed. Modeling each drug is not sufficient when micro-services already exist that provide a searchable interface for drugs with all crucial information. Therefore, the drug type is modeled as JSON types attribute while the `uiReference` external enumeration is used to render the JSON attribute as a searchable drop-down list comparable with a native human-readable enumeration attribute. The `uiReference`

provides additional information such as the URL, where to request the searched drugs. After a drug is selected, only the drug identifier and label are stored JSON-encoded. Furthermore, the care professionals declare the dosage as a simple string, the start date, the end date, the frequency including the unit, a maximum delay for taking, and an optional long text comment. While the drug prescription is active, the case execution engine receives time series data showing the patient's drug-taking behavior. The measurement attribute modeled as JSON typed attribute is interpreted as a diagram using the `uiReference`.

- **Patient Questionnaire** A patient questionnaire allows care professionals to ask patients to complete different questionnaires repeatedly on a regular basis. Similar to the monitoring prescription, professionals provide information such as the questionnaire type to be filled in including SF12, EQ5D, and many more. Furthermore, professionals declare a start date, end date, the frequency including the unit, a maximum delay, the time slot to complete it, and maximum repetitions per day. While the patient questionnaire is active, the case execution engine receives the completed questionnaires as JSON time series data which is interpreted with the `uiReference` as patient questionnaire data. Professionals are especially interested in the changes in the patient's answers to evaluate the treatment. The special rendering for patient questionnaires is needed to show multiple answered questionnaires embedded within one task. Natively, a `HumanTask` followed by many `AutomatedTasks` would be conceptually similar, but the usability for care professionals would decline significantly. Not to be confused with the `DualTask` concept where exactly one `AutomatedTask` follows.
- **Physical Activity** A physical activity prescription allows care professionals to declare how many steps a patient should walk daily and to monitor a patient's behavior. Therefore, parameters such as the start and end date and the number of daily steps allow tailoring the prescription. Expected low-, medium- and high-level activity times are customizable.
- **Simple Task** A simple task allows care professionals to assign simple predefine tasks to patients such as reading a book or walking outside slowly. Complementary to that, the simple task type other allows the ad-hoc declaration of a non-predefined simple task. Besides the task type, the frequency, maximum delay, start date, end date, time slot, the repetitions per day, and a comment can be declared. Similar to the drug prescription, the professional receives feedback with a line diagram when the simple task was accomplished.
- **Advice** An advice allows care professionals to provide patient-centered instructions. The professional provides a short title, an expire date, and an explanation for the patient. As patient feedback, the care professional receives the visualization on the patient mobile application.

The case study templates differ due to local site-specific model requirements. Table 6.2 summarizes which models are applied in each case study, while Table A.2 provides a more comprehensive summary concerning the detail enumeration options. Even though all sites use the blood pressure monitoring prescription, there are slight differences. All case studies use only one workplan stage except Lleida CS2 which splits the workplan stage in pre- and post-hospitalization and allows monitoring in both stages. Additionally, Lleida separates certain monitoring prescription options to enable defining more precise default values. On the lower level, many site-specific model customizations are applied. I.e., the measurement time slot options are adapted according to the local culture.

	Groningen		Tel Aviv		Lleida	
	CS1	**CS2**	**CS1**	**CS2**	**CS1**	**CS2**
Monitoring Prescription	◐	●	◐	◐	◐	◐
– Blood Pressure	●	●	●	●	●	●
– Body Temperature	●	●	○	○	○	○
– Weight	○	●	○	○	●	○
– Heart Rate	○	●	○	○	●	○
– 2 more specific ...	○	○	○	○	●	○
Drug Prescription	◐	◐	◐	◐	◐	◐
Physical Activity	●	●	●	●	●	●
Patient Questionnaire	◐	◐	◐	◐	◐	◐
– SF-12	●	●	●	●	○	○
– EQ5D	●	●	●	●	○	○
– AHS post-chirurgical	○	●	○	○	●	●
– S-LANSS	○	○	○	○	○	●
– 18 more specific ...	○	○	●	●	○	○
Simple Task	○	○	●	●	○	○
– Read book or newspaper	○	○	●	●	○	○
– Walk slowly outside	○	○	●	●	○	○
– 23 more specific ...	○	○	●	●	○	○
– Other	○	○	●	●	○	○
Advice	●	●	●	●	●	●

○ not applied ◐ partly applied ● fully applied

Table 6.2.: `DualTask` system integration and orchestration models each representing a corresponding micro-service within the SMS.

6.5.3. Iterative Case Template Modeling Process

Initial observations regarding the case template modeling processes have been described by Michel et al. (2019). In the following, we present an extended observation time scope with slightly adapted naming. Clinical processes are knowledge-intensive processes in a complex environment that cannot be designed completely upfront. We analyzed several key-factors that lead to evolutionary case template changes:

- **User-introduced iterative adaptations** are typically small iterative improvements such as corrections of translation labels, adding additional descriptions for task parameters to explain crucial contextual information, or changing task footnotes to provide hints or to address the copyrights sufficiently.
- **Practice-introduced adaptions** stem from clinical practice such as removing a sentry definition to enable simultaneous work on multiple stages, changing the default values, or to apply modeling best practices from other case studies. I.e., the summary page model contains a dynamically generated visual body representation initially modeled for Lleida CS2 that was adapted and applied in Lleida CS1 and Groningen CS2.
- **Feature introduced adaptations** occur when a new version of the case-execution engine is deployed that supports new modeling capabilities. During the CONNECARE implementation studies, multiple new case template modeling capabilities such as custom data representation to override the default data representation became available and led to

model changes. Rarely, newly introduced features affect the modeling syntax of existing features, which requires an adaptation of all existing case templates before importing.

A case template is declared with an XML schema that is imported into the case execution engine to become executable. All case templates, technically represented with XML files, are managed in a git repository that allows tracking changes including meta-data such as modification date and editor. To apply a case template change, a modeler performs modification on the XML file, commits and pushes the changed file into the repository. Our analysis is based on a git log command to receive all changes of a certain model definition file. Since the XML case template schema was not stable enough before October 2017 due to new feature development, we only consider changes after that date. The XML structure is parsed to analyze a case template while XML files that cannot be parsed according to the schema are ignored. Schema adaptions caused by the continuously extended case execution engine are considered while parsing. In total, 710 commits were found that modified a case template, whereas 631 valid commits were analyzed further. In average, 11.2 percent of the commits of each case study could not be parsed and have been ignored. Based on the git log command, we implemented an initial script that parses a case template file and counts all model elements and their relations. All results are stored in a large spreadsheet with 175 KPIs per commit and the related meta-data including commit message, date, and author. We explored the pre-processed change sets manually and derived the following classification categories:

- **Structural Change** is a modification that adds or removes core workflow elements such as a `StageDefinition`, `HumanTaskDefinition`, or a `DualTaskDefinition`. E.g., the addition of a new `HumanTask` to calculate the BMI of a patient.
- **Adaptation Change** is a modification that adds or removes model elements that were not considered for structural changes. E.g., the addition of an additional `TaskParamDefinition` for a task, the addition of a `SentryDefinition` as a task pre-condition, or the addition of a `SummarySectionDefinition`.
- **Simple Change** is a modification that adapts string labels which do not modify the structure at all. E.g., the fix of a wrong translation of a label, or the addition of an additional description for a linked `AttributeDefinition` of a task parameter to clarify the meaning for the care professional.

The presented classification categories are ordered by change complexity, beginning with the most complex. A change may belong to multiple classification categories but we only assign the label of the most complex category. Figure 6.13 illustrates the case template evolution of each case study based on the analyzed and classified commits. All commits between October 2017 and mid May 2019 were considered. The case studies are grouped according to hospital sites and show the number of classified commits over time. Vertical dotted purple lines indicate model deployments on the production environment. In total, 44 case templates were deployed. The CONNECARE implementation studies first focused on the case template modeling of CS2 to gain experience on all sites and then proceeded with modeling CS1. The case templates of CS1 have been elaborated much faster due to experience and the re-usability of case template elements. Within the early stage of the CONNECARE implementation studies, newly introduced case execution engine features were mostly practical tested in Groningen CS2 which implies case template changes were applied slightly earlier. Later, new features were applied in Lleida first.

Comparing the structural changes across sites indicates more structural changes were applied in Lleida. Several generic task definitions were decomposed into multiple task definition to support specific default values which caused additional structural changes. All case templates tend to stabilize up to a certain degree over time. However, we noticed changes are continuously required to support organizational needs.

Figure 6.14 illustrates the classified commits for each modeler ordered by the number of total commits. For each case study, one modeler is responsible for coordinating the communication with the clinical partners which is also indicated in the summary table on the right. Due to organizational changes, the responsibilities changed multiple times during the project. First, modeler E was responsible for the Lleida case studies which was taken over by Modeler A. In the past, Modeler B and Modeler D have been responsible for the Groningen and Tel Aviv case studies which are now under responsibility of Modeler C. The numbers indicate three main contributors who were responsible for approximately 91 percent of all commits. Modeler A performed 247 commits or 39 percent, modeler B performed 203 commits or 32 percent, and modeler C performed 126 commits or 20 percent. Case template modeling is an iterative process which requires multiple feedback loops to create tailored models according to the case study needs. Therefore, the current organizational setup tries to distribute the case template changes equally among the partners who coordinate the changes directly with the clinical partners. Arising conceptual issues are centrally coordinated by modeler A. Typically, one case study is used to demonstrate a proper meta-model solution. When required, this is then applied by multiple modelers on all sites. The tailored case templates are a critical success factor strongly influencing the system acceptance. Therefore, a reasonable modeling process is essential.

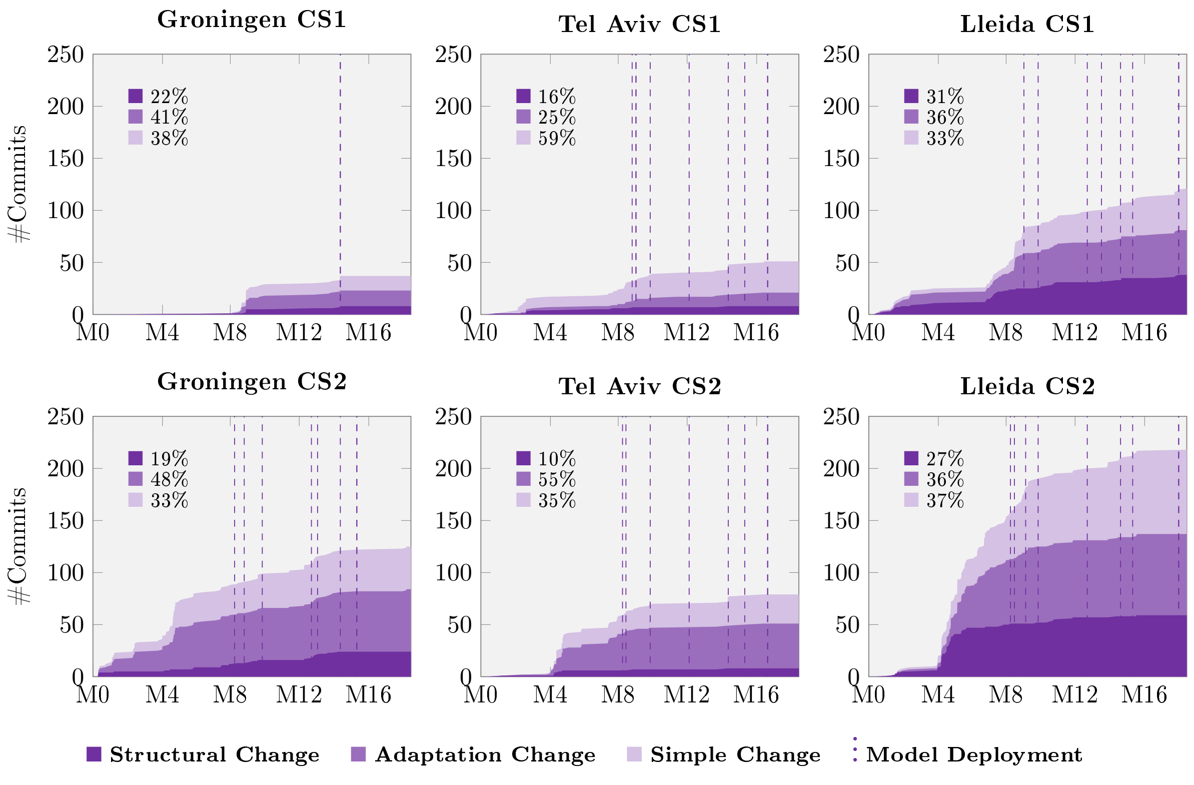
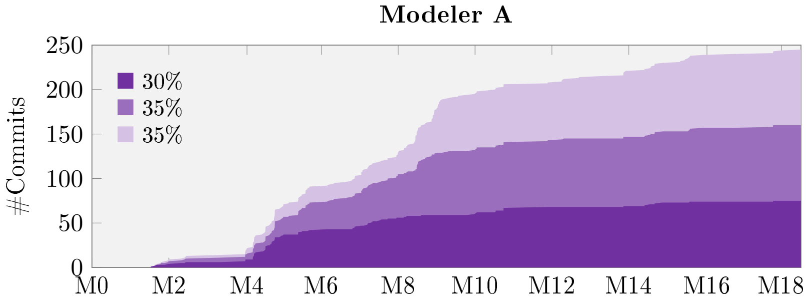
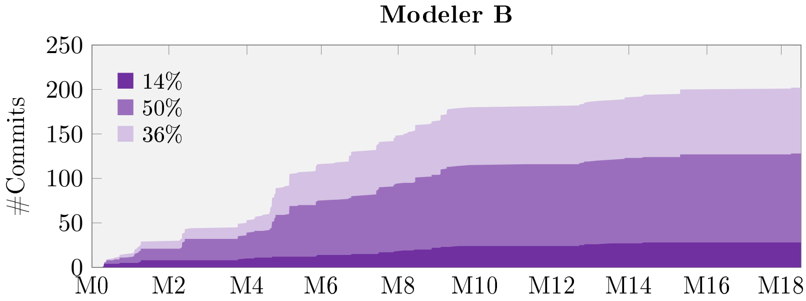


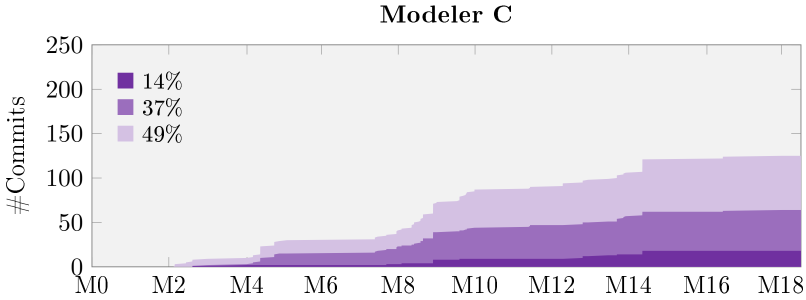
Figure 6.13.: Model evolution based on classified commits comparable across case studies.



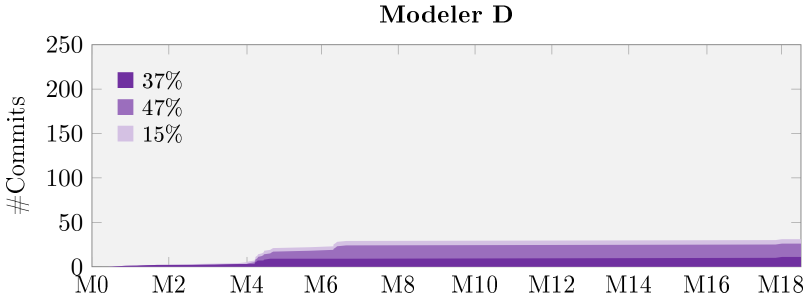
Location	CS	■	▨	□
Groningen	CS1	-	-	-
	CS2	-	1	-
Tel Aviv	CS1	-	-	-
	CS2	-	1	-
Lleida	CS1	27	34	34
	CS2	48	49	53
∑		75	85	87



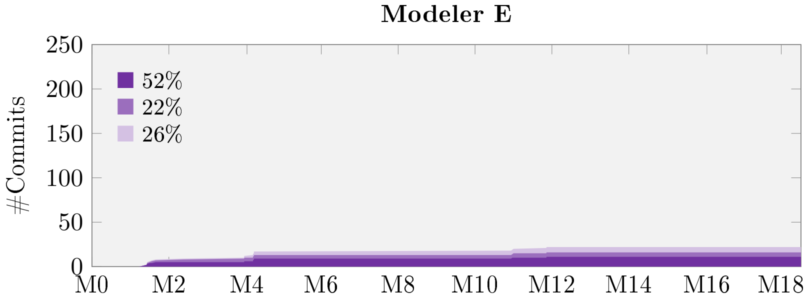
Location	CS	■	▨	□
Groningen	CS1	-	3	-
	CS2	18	51	30
Tel Aviv	CS1	1	5	5
	CS2	3	16	12
Lleida	CS1	2	4	4
	CS2	4	22	23
∑		28	101	74



Location	CS	■	▨	□
Groningen	CS1	8	12	14
	CS2	5	3	9
Tel Aviv	CS1	4	8	24
	CS2	1	23	15
Lleida	CS1	-	-	-
	CS2	-	-	-
∑		18	46	62



Location	CS	■	▨	□
Groningen	CS1	-	-	-
	CS2	1	5	2
Tel Aviv	CS1	3	-	1
	CS2	4	3	1
Lleida	CS1	3	-	-
	CS2	1	7	1
∑		12	15	5



Location	CS	■	▨	□
Groningen	CS1	-	-	-
	CS2	-	-	-
Tel Aviv	CS1	-	-	-
	CS2	-	-	-
Lleida	CS1	6	5	2
	CS2	6	0	4
∑		12	5	6

■ **Structural Change** ▨ **Adaptation Change** □ **Simple Change**

Figure 6.14.: Evolutionary contributions of individual modelers, including a summary table indicating case study specific contributions.

6.6. CONNECARE Implementation Studies Case Execution

This Section focuses on aspects regarding the case execution. Instantiated cases are summarized considering sites and case studies in Section 6.6.1. Additionally, the timeline regarding the number of instantiated cases is presented, while the availability of new case template versions is shown in context. To gain insights regarding the case execution behavior, process discovery techniques are applied as described in Section 6.6.2. The discovery methods are applied with a focus on flexible process adaptation during run-time as elaborated in Section 6.6.3. Furthermore, aspects regarding communication and coordination are elaborated in Section 6.6.4. The case team related behavior is described in Section 6.6.5. Finally, a comprehensive high-level case execution metrics summary is provided in Section 6.6.6.

6.6.1. Instantiated Cases

During the implementation studies, chronical patients are treated over a period of approximately 12 months from May 2018 until mid May 2019. A patient treatment begins with the case template instantiation that evolves to a patient-treatment-specific case instance. We carefully analyzed the production system and noticed test patients who are primarily used to approve new system release features working as expected across all systems. To clearly identify the actual patient cases, the site-specific responsible clinical professionals annotated those. Consequently, 136 test cases are identified and ignored for further analysis which is approximately 37 percent of all available cases within the production system.

Table 6.3 summarizes all instantiated cases representing actual patients. We distinguished patients according to the clinical sites Groningen, Tel Aviv, and Lleida, participating in the case studies. The average patient age is 70.6 years, whereas the median is 71 years. The oldest patient is a 96-year old man participating in case study one located in Lleida, whereas the youngest patient is a 20-year old woman participating in case study one located in Groningen. The gender distribution is largely even except for case study two in Groningen with 72 percent male patients. Additionally, privacy considerations allow declaring one's gender as unknown, which concerns case studies one and two in Groningen with 4 percent and 5 percent, respectively. With 57 patients, case study one located in Tel Aviv is the largest. Across all sites and case studies, 232 patients were treated with instantiated case templates tailored with run-time planning to patients' individual needs.

Location	**Case Study**	**Patient**			
		⌀ Age	Male	Female	Cases
Groningen (NL)	CS1	58	40%	56%	25
	CS2	73	72%	23%	43
Tel Aviv (IL)	CS1	67	42%	58%	57
	CS2	65	47%	53%	32
Lleida (ES)	CS1	83	56%	44%	39
	CS2	73	42%	58%	36
Total		**71**	**50%**	**48%**	**232**

Table 6.3.: Instantiated cases across sites.

Figure 6.15 visually illustrates the patient inclusion or the technically named case instantiation in chronological order. The clinical sites are distinguished with columns and the case studies with rows. Each case study is depicted with a yellow colored diagram illustrating the time passed on the x-axis and the number of included patients on the y-axis. Aggregated site and case study information are visualized on the right side and at the bottom featuring an enlarged y-scale. New iteratively imported case templates are indicated with purple vertical dashed lines. After a successful case template import, the instantiation option of the previous versions is automatically disabled to enforce using the latest case template declaration and to facilitate an iterative improvement process. The average inclusion rate across all sites and case studies is approximately 4.5 patients per week. Local site-specific inclusion rates slightly differ as they primarily depend on suitable patients.

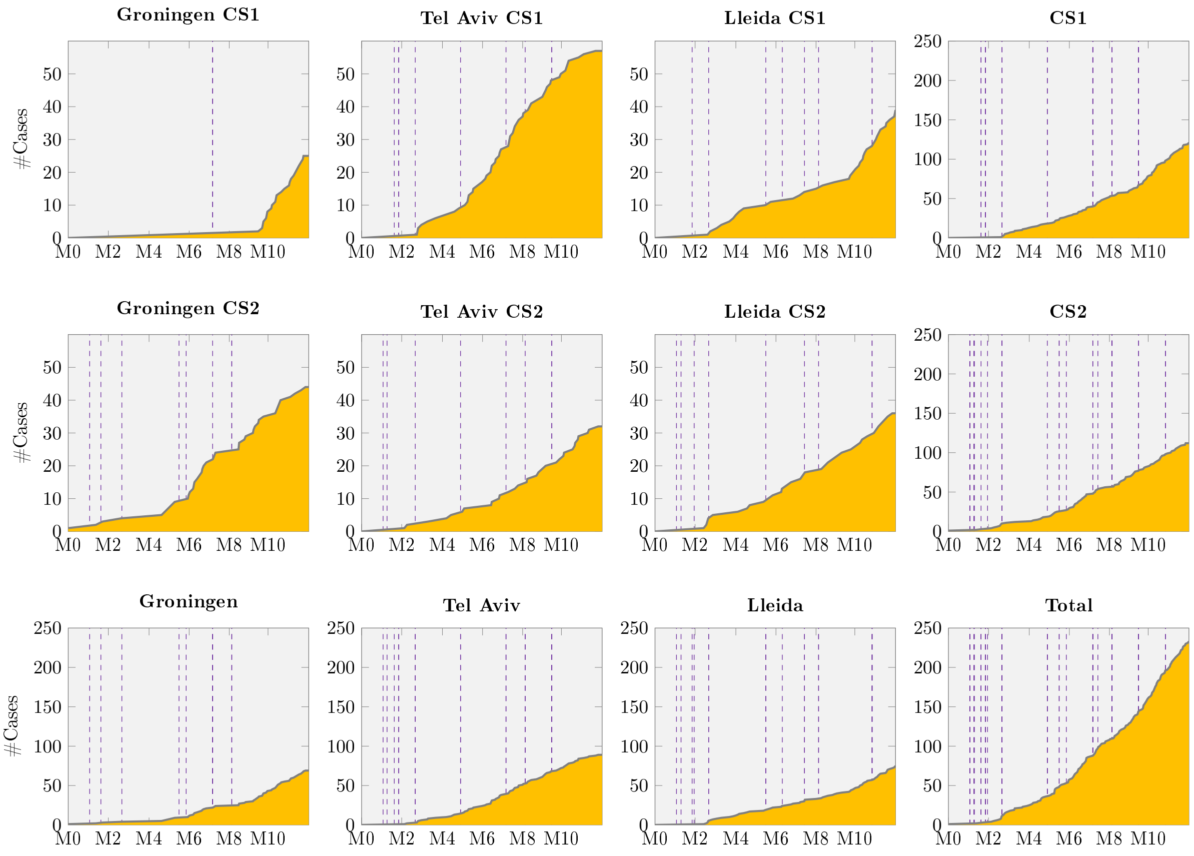


Figure 6.15.: Included patients or technically named instantiated cases are visualized in chronological order. The accumulated number of cases is highlighted in yellow and the deployed case templates are indicated with a dashed purple line.

6.6.2. Process Discovery to Analyze Case Execution Behavior

Characteristically, knowledge-intensive processes cannot be determined entirely during model design-time. Therefore, the case templates allow declaring certain restrictions and provide the required run-time flexibility. Communication and coordination aspects, as well as role assignments, may influence run-time planning. Therefore, we intensively advised a master thesis that focused on discovering clinical pathways of an adaptive integrated care environment (Bönisch, 2019). In the following, we present the summarized and extended results from this thesis.

Figure 6.16 illustrates the conceptual discovery pipeline. Case templates are imported into the case execution engine to become executable. Care professionals instantiate cases when including new patients into the case studies. During the patient treatment, care professionals perform activities such as manually activating a task, completing a task, terminating a task, correcting a task, acknowledging a received alert, sending a team message, sending a patient message, acknowledging an alert or notification, editing notes, granting case access rights, editing case access rights, revoking case access rights, and editing case roles to name most crucial activities. Case execution activities are mostly captured based on an API log recording all SACM inbound requests (cf. Section 6.3). Due to client pre-fetching mechanics, GET requests are ignored and only modifying requests are considered. The execution log with the activities allows discovering the case execution behavior. Conceptually, the discovery distinguishes three abstraction levels or views to provide adequate information. The low-level view contains activities on stage-level, i.e., completing a task. Similarly, the case-level view provides a more comprehensive summary, including the messaging, alerts, and notes activities. The system-level view provides the highest abstraction level for all occurred activities. In the following, we will focus on activities on the stage- and case-level. The analyses do not distinguish different case template versions and consider the actual number of executed cases within a case study.

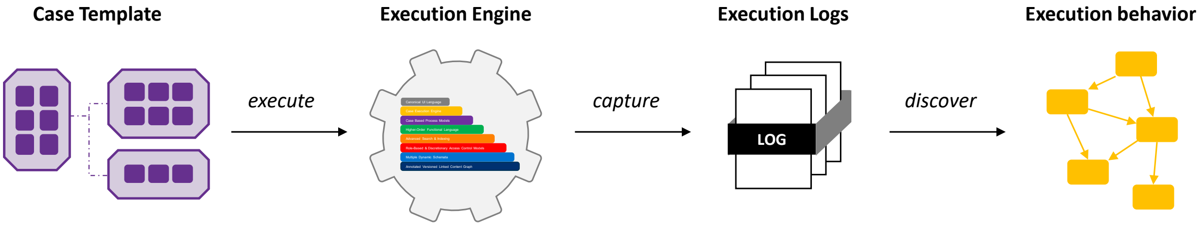

Case Template
Execution Engine
Execution Logs
Execution behavior
execute
capture
discover
Canonical UI Language
Case Execution Engine
Case Based Process Models
Higher-Order Functional Language
Advanced Search & Indexing
Role-Based & Discretionary Access Control Models
Multiple Dynamic Schemata
Annotated Versioned Linked Content Graph
LOG

Figure 6.16.: Conceptual process discovery pipeline to analyze case execution behavior, adapted from Bönisch (2019).

Figure 6.17 illustrates the technical pipeline separated into online processing with the elastic stack and offline processing with Disco. The elastic stack combines the search engine Elasticsearch to index documents, Logstash to pre-process and ship logs, and Kibana to explore raw data visually with dashboards. The elastic stack enables nearly real-time log processing and is deployed as a customized docker container (cf. Section 6.4, sacm.analytics). Disco is used for the final process mining step to visualize the different execution graphs and enables the abstraction of results to potentially conceptualize the execution behavior.

The SACM execution engine provides REST-based API interfaces to enable interaction with the corresponding frontend application and neighborhood systems. All inbound HTTP requests are logged and stored in an SQL-compatible database. An access log entry contains structured meta-data such as the request method, the request URL, the abstract request pattern, the final request state, and several more fields for analytical purposes. In total, approximately 13 million raw access logs are captured, most entries whereof represent the read-only requests. Conceptually, the patient messaging is not part of the SACM case execution engine and not contained within the default access logs. Patient messages are an essential part of the CONNECARE concept. Therefore, the production database is used in addition to extract required meta-data comparable with access logs.

Logstash pipelines are decomposed into several small pipelines to increase the reusability and to ensure the maintainability. In one-minute intervals, a JDBC connector requests the latest access logs from the database. The latest access logs are received using a timestamp or a primary identifier of a previously imported access log to process only the newly created access logs. Within the pre-processing step, identifiers contained within the URL are extracted and the related activity is determined using the URL pattern. Based on the access log, certain information cannot be determined which varies depending on the actual API endpoint. Therefore, an additional lookup step is required to enrich the access log with information such as the case context and the related model definition. Afterwards, access log specific post-processing is accomplished depending on the primary data source to unify data formats. The results are stored for the log and pattern inspection with Kibana and are further processed for process mining purposes. All previous steps increase the information density and improve quality while the access logs are transformed into event logs used for process mining within the next steps. A filter ensures that the case represents an actual patient, that the used API endpoint is relevant for process mining, and that the related request was completed successfully with an HTTP 2xx status code. An access log entry still contains many properties irrelevant for process mining. Therefore, a whitelist is applied to reduce the noise and memory footprint. The Logstash translate filters are applied to pseudonymize identifiers with human-readable random strings. Additionally, filters are applied to improve naming consistory. Certain events cannot be captured sufficiently in the access log. For performance reasons, the raw access log contains payload data from the request

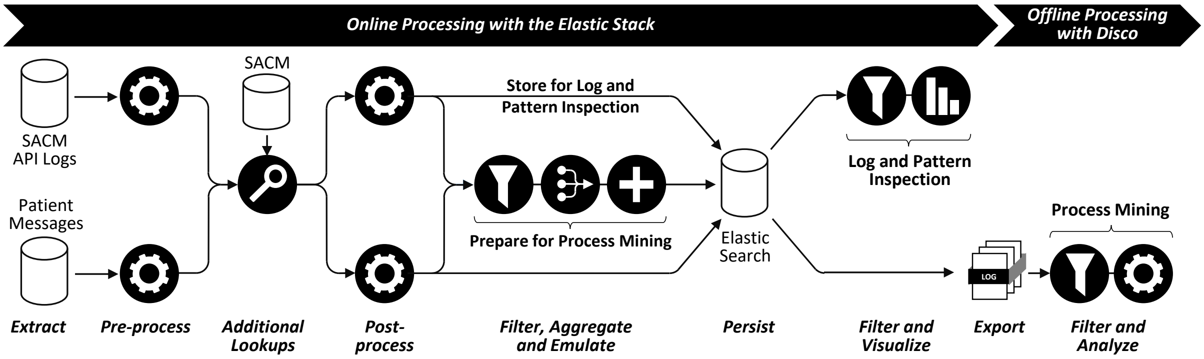

Online Processing with the Elastic Stack
Offline Processing with Disco
SACM API Logs
Patient Messages
SACM
Store for Log and Pattern Inspection
Prepare for Process Mining
Elastic Search
Log and Pattern Inspection
Process Mining
LOG
Extract
Pre-process
Additional Lookups
Post-process
Filter, Aggregate and Emulate
Persist
Filter and Visualize
Export
Filter and Analyze

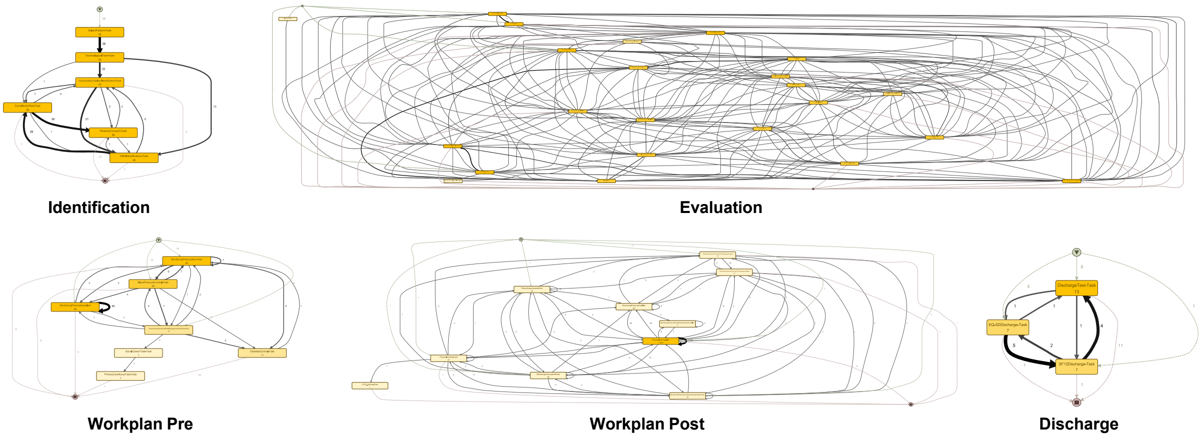
Figure 6.17.: Technical process discovery pipeline to analyze case execution behavior adapted and extended from Bönisch (2019).

and captures the response body only for unsuccessful responses. Instantiating a new case definition returns the resulting case within the response body, including the identifier. Similarly, the problem occurs for events internally triggered in the engine. Therefore, important and not sufficiently logged events are simulated based on timestamps from the production database using Logstash pipelines. Finally, the resulting data is stored in the elastic search index.

With the enriched information stored in the elastic search index, the log and pattern inspection can be performed with Kibana. Custom configured visualizations help identify simple usage patterns and help detect possibly occurring anomalies, as illustrated in Figure A.11. If required, additional dashboards connected to the elastic search can be configured quickly. Preprocessed data required for process mining is exported for further processing with Disco, which is a specialized process discovery software. Disco calculates and visualizes several metrics that are helpful to describe an executed process. The following sections focus on process discovery results for the case studies performed within the CONNECARE project.

6.6.3. Flexible Process Adaptation at Run-Time

First, the observed execution characteristics are described as a context for the flexible process adaptation or alternatively named run-time planning. As elaborated in Section 6.6.2, visual case execution models are created on a case and stage abstraction level. Figure 6.18 visualize the stage abstraction level indicating the execution activities for each stage in Lleida CS2 (cf. Section A.5 for full-size visualizations of all case studies). Activities, or – more precisely expressed – mostly task completions are represented as nodes. A darker color indicates a higher occurrence frequency, which is also explicitly expressed with the number below the activity name. Subsequent activities are indicated with edges. Thicker edges indicate a higher occurrence frequency, which is also explicitly expressed with a number next to the edge. Corresponding to the stage execution visualization, Table 6.4 (cf. Section A.5 for Groningen and Tel Aviv metrics) illustrates crucial execution metrics on the stage- and case-levels with higher aggregated activities. I.e., on the case-level, all task completion activities of a stage are aggregated to one abstract task completion activity. A similar pattern is applied for all activity types. An instantiated case does not necessarily need to perform all declared stages. Therefore, the number of cases expresses how often the stage occurred during execution. The number of activities and the number of overall paths are basic metrics to describe the executed cases. Relative metrics enable the comparison of different stages and case executions. The mean number of paths per activity counts all in- and outbound paths that belong to an activity. The number of process variants is calculated based on an equal activity occurrence order only considering the activity name. Derived from it, the maximum share of a case per variant and the mean share of manual activated tasks are calculated to identify potentially highly knowledge-intensive executions. Up to a certain degree, the mean share of manually activated tasks depends on the declarations on the case template. When tasks are repeatable, the number of actually executed repetitions influences this metric.


Identification

Evaluation

Workplan Pre

Workplan Post

Discharge

Figure 6.18.: Case stage execution in Lleida CS2.

	CS1					CS2					
Indicator	**Identification**	**Evaluation**	**Workplan**	**Discharge**	**Case Level**	**Identification**	**Evaluation**	**Workplan Pre**	**Workplan Post**	**Discharge**	**Case Level**
Number of cases	38	39	39	14	39	36	35	34	29	13	36
Number of activities	5	22	12	3	10	6	23	7	10	3	12
Number of overall paths	8	187	69	5	51	15	206	19	55	6	77
Mean number of paths per activity	3.2	17	10.7	3.3	9.4	5	17.9	5.1	10.3	4	12
Number of process variants	3	39	39	5	39	7	24	35	29	5	36
Maximum share of cases per variant	79%	3%	3%	29%	3%	58%	3%	18%	3%	46%	3%
Mean share of manual activated tasks	0%	15%	86%	0%	28%	0%	5%	20%	67%	0%	15%

Table 6.4.: Case execution characteristics and run-time planning in Lleida.

	Groningen		**Tel Aviv**		**Lleida**	
	CS1	**CS2**	**CS1**	**CS2**	**CS1**	**CS2**
Number of cases	25	43	57	32	39	36
Number of activities	6	8	9	9	10	12
Number of overall paths	14	20	44	38	51	77
Mean number of paths per activity	4	4.4	9.1	7.8	9.4	12
Mean number of paths per alert activity	7	8	11	10	14	12.5
Mean number of paths per message activity	5	3	12	10.5	13	18
Number of process variants	19	42	56	32	39	36
Maximum share of cases per variant	16%	5%	4%	3%	3%	3%
Mean share of manual activated tasks	7%	24%	12%	17%	28%	15%

Table 6.5.: Case execution characteristics and run-time planning across sites.

The manual task activation, or alternatively named run-time planning, only occurred in Groningen and Tel Aviv in the workplan stage where all tasks were manually activated. Accordingly, the maximum share of cases per variant is lower than in all other stages. In Lleida, the evaluation and workplan stages contains manual task activations. The evaluation stages in CS1 and CS2 contain a small share of manual task activations and the maximum share of cases per variant is very low. The workplan stages partly consist of manually activated tasks. In Lleida CS2, the workplan stage is separated into a workplan pre-stage with 18 percent maximum share of cases per variant and 20 percent manually activated tasks and into a workplan post-stage with 3 percent and 67 percent, respectively. Table 6.5 summarizes the metrics on case-level across all study sites. An additional metric regarding the mean number of paths per alert or messages per activity indicates a higher variance regarding predecessor and successor activities. Based on the observations made during the case studies and based on the analyzed case execution data, the workplan stages most actively use the manual task activation and run-time planning.

6.6.4. Communication and Coordination Behavior

Communication and coordination activities are essential parts of patient treatment. Therefore, we analyzed the usage behavior regarding the case alerting or according to frontend terminology notifications, case messaging, and case notes features. Table 6.6 illustrates relevant alert metrics across all case studies. Overall, 16,912 alerts occurred and most of them were domain-specific and custom generated alerts from the SMS micro-services that push those alerts into the workflow engine. As expected, the number of correct alerts that occur when a completed or terminated task is modified is relatively low. Within our test environment, many error alerts occurred due to typical interoperability problems between the declared case templates and the related hard-wired micro-services that are connected for integration. Errors most likely occur when hook definitions are not successfully executed, or when, the HTTP response status code is not 2xx. Considering that the maintenance of the SACM and SMS systems is synchronized in the production environment, no errors occur. However, this is very useful for testing and most

	Groningen		**Tel Aviv**		**Lleida**	
	CS1	**CS2**	**CS1**	**CS2**	**CS1**	**CS2**
Occurred alerts	1451	4344	3574	2498	2960	2085
├ error type	0	0	0	0	0	0
├ correct type	5	16	71	68	124	92
└ custom type	1446	4328	3503	2430	2836	1993
Average words/alert	7	7	8	7	11	11
Percentage of acknowledged alerts	27%	99.5%	92.2%	70.9%	99.9%	99.8%
Average days until acknowledged	5.3	5.5	8.4	14.8	0.8	0.6
Percentage of cases with occurred alerts	100%	98%	84%	100%	97%	92%
└ Average alerts/case	58	103	74	78	78	63

Table 6.6.: Alerts summary across sites.

likely occurs when orchestrating foreign services. In practice, the alerts must be short and meaningful to ensure the best usability. Long alert messages need more attention from care professionals and due to readability issues, those are shortened on the dashboard. Across the different case studies, an alert contains on average 7 to 11 words, mostly depending on the distribution of the different custom alerts. Occurred alerts provide the possibility to be marked as acknowledged, which removes them from the personalized dashboard and does not highlight them on the task page anymore. Except for CS1 in Groningen and CS2 in Tel Aviv, nearly all alerts have been acknowledged. Considering all acknowledged alerts, the average time for an alert to be acknowledged varies significantly. Lleida acknowledges alerts within less than a day, whereas in Tel Aviv CS2, the average time was approximately 15 days. Alerts occurred in nearly all cases with the exception of Tel Aviv CS1 where only 84 percent of the cases contain alerts. The average number of alerts per case across all sites is between 58 in Groningen CS1 and 103 in Groningen CS2.

The messaging usage behavior is distinguished into case team messaging where the patient is not involved and case-patient messaging where the team communicates with the patient. Table 6.7 illustrates relevant indicators to compare the case team messaging across all case studies at the top and the case-patient messaging at the bottom. In total, 1,474 team messages and 2,047 patient messages were sent. Groningen CS1 did not use the team messaging feature at all. The percentage of cases using the case team messages indicates that apart from Groningen CS1 and CS2, the case team messaging feature was actively used in most cases. During feedback meetings, we noticed that in Tel Aviv, the case team messaging feature was also used to document patient treatments. Team messages were authored by two to three care professionals except at Groningen. On average, a team message contains approximately 29 words across all sites. The usage of case-patient messaging slightly differs. The feature is used at all sites, whereas the percentage of cases using patient messaging is comparatively lower. The reasons might differ, but we need to consider that not all elderly patients are able to use the messaging feature, which might be one reason. In comparison, the average number of messages per case is much higher, especially in Lleida.

		Groningen		**Tel Aviv**		**Lleida**	
		CS1	**CS2**	**CS1**	**CS2**	**CS1**	**CS2**
Case Team	Number of messages	-	9	578	431	264	192
	Average words/message	-	36	28	23	37	34
	Percentage cases using messages	-	14%	82%	100%	92%	94%
	├ Average messages/case	-	1.5	12.3	13.5	7.3	5.6
	└ Average unique authors/case	-	1.0	1.8	2.9	2.3	1.9
Case Patient	Number of messages	106	6	375	236	741	583
	Average words/message	39	5	12	12	16	13
	Percentage cases using messages	64%	12%	75%	84%	79%	81%
	├ Average messages/case	6.6	1.2	8.7	8.7	23.9	20.1
	└ Average unique authors/case	1.4	1.2	2	2.6	2.5	2

Table 6.7.: Messages summary across sites.

The case notes feature was mostly used in Lleida to provide a space for information exchange between hospital care roles and primary care roles. Therefore, the case templates contained a predefined case notes structure modeled with an HTML table to contain two columns, one for the hospital care roles and one for the primary care roles. Each row represents the cosponsoring roles at hospital care and primary care. While the case is progressing, the notes lead to documentation summarizing the primary care and hospital care activities according to the involved roles (cf. screen in Figure 5.1.14). The average number of words used on the notes page is approximately between 131 in Lleida CS1 and 139 in Lleida CS2. The other sites occasionally used the notes feature for unstructured documentation purposes.

6.6.5. Case Team Members and Roles Behavior

During the case template modeling phase, case team roles are declared implicitly within the case template. Attributes referencing users who are linked as task owners implicitly become a case team role. User referencing attributes allow declaring constraints such as the user must be part of a set of groups to ensure only suitable users can be assigned. During the project, the case templates were enhanced multiple times to declare a more sophisticated case role model. During run-time, each user assigned to a case team role automatically becomes a case team member. The case team members represent a set of users that has either read or write access for the related case. Figure 6.19 illustrates a conceptual model with a focus on the case team management to explain the context of the table presented in the next paragraph. Purple and light blue colored elements are part of the case template created during the modeling phase, while yellow and dark blue colored elements are created during the run-time phase. Users and groups are typically pre-existing and reused within multiple case templates and case instances.

The case team concept enables working collaboratively on dedicated cases. Table 6.8 presents crucial facts regarding the case team members and the role management per case. The case templates in Lleida contain most role declarations. On average, the Lleida CS2 case template declared 11.3 roles across all case template versions. In both Groningen case studies, only one role was declared. In Lleida, the maximum number of team roles is slightly higher as the average,

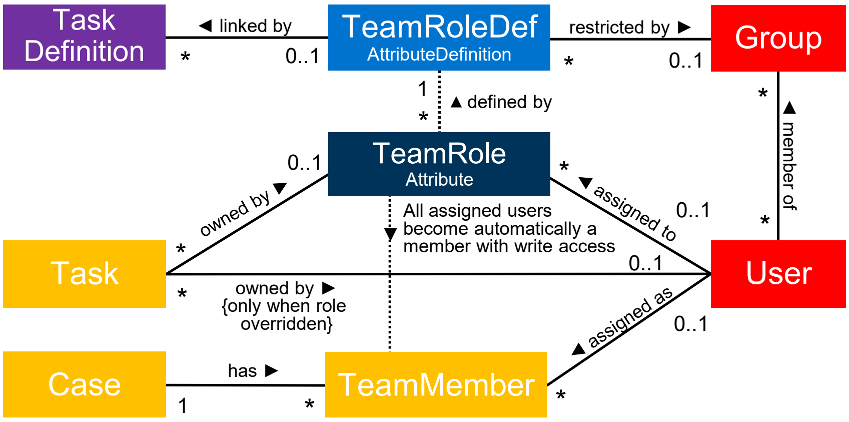


Figure 6.19.: Conceptual case team management during modeling and run-time phase.

	Groningen		Tel Aviv		Lleida	
	CS1	**CS2**	**CS1**	**CS2**	**CS1**	**CS2**
Average team members with at least read access	1	1.9	2.3	3.8	6.1	8
Max. team members with at least read access	1	6	4	5	12	14
Average team members with at least write access	1	1.8	2.3	3.8	5.8	7.8
Max. team members with at least write access	1	3	4	5	7	10
Average team roles	1	1	2	3	5.1	11.3
Max. team roles	1	1	2	3	6	12
Average add or modify team member operations	2	2.4	8	10.2	6.9	9
Average delete team member operation	0	0	0	0	0	0
Average modify team role operations	2	1.9	7.6	9.9	6.8	9

Table 6.8.: Case team members and roles summary across sites.

which indicates that the model evolved over time. When assigning a user to a declared role, write access is granted automatically. Therefore, the number of declared roles influences the number of users who have case access rights. In Lleida CS2 and Tel Aviv CS2, the average number of team members with write access was slightly higher, which could be an indication that an additional role might be required. In Lleida CS2, the average number of declared roles was higher than the average number of team members with write access per case, which is explained by multiple role assignments per user. During run-time, team roles were assigned and team members were added and deleted. These operations are triggered implicitly when a linked attribute is changed or explicitly when a user manually performs an action on the team page. Especially in Tel Aviv, the number of team role modifications exceeds the number of declared team roles which means specific roles were changed multiple times. A similar pattern occurs for the team member operations. Until the end of the data gathering period, the revoke team member access operation was not performed. To conclude, in Lleida the system was used more collaboratively than at all other sites. In Groningen CS1, collaborative features were not used.

6.6.6. Summary of Case Execution Behavior

In total, 232 patients across six case studies were treated using the ACM4IC approach. Table 6.9 provides highly aggregated metrics regarding the case execution behavior (cf. more detailed metrics in Section 6.6). The number of instantiated cases, or in medical terminology treated patients, is reasonably distributed across the case studies while Tel Aviv CS2 represents the maximum with 57 instantiated cases and Groningen CS1 the minimum with 25 instantiated cases.

Integrated communication and coordination functionalities are crucial factors to support integrated care. The mean number of team roles declared within the case templates shows a high variance between the hospital sites and the case studies. Similarly, the mean number of team members per case having at least read access indicates a correlated distribution. The vast differences result from diverse organizational settings. Lleida indicates the highest collaboration among case professionals. Comparing the deviation between those two metrics, most case studies have more team members than declared team roles except Lleida CS2 where particular roles are not used in all case instances. On average across all sites, each case has between 58 and 101 occurred alerts. Nearly all occurred alerts result from integration tasks such as monitoring prescriptions where declared thresholds are exceeded. The average time until an alert is acknowledged varies significantly depending on the hospital site. Overall, the messaging feature is widely used and the message metric aggregates both professional-to-professional and professional-to-patient messaging.

The required degree of structure depends on the stage context. While a case inclusion stage typically does not require flexible run-time planning, workplan stages that represent a patient-specific treatment plan require a significant amount of run-time planning. Workplan stages typically have a lower maximum share of cases per execution variant compared to all other case stages. Similarly, manual task activations indicate knowledge-intensive stages that highly depend on individually created patient treatments.

	Groningen		**Tel Aviv**		**Lleida**	
	CS1	**CS2**	**CS1**	**CS2**	**CS1**	**CS2**
Number of cases	25	43	57	32	39	36
Mean number of team members per cases*	1	1.9	2.3	3.8	6.1	8
Mean number of team roles	1	1	2	3	5.1	11.3
Mean number of alerts per cases	58	101	62.7	78.1	75.9	57.9
Mean number of messages per cases	4.2	0.3	16.7	20.8	25.8	21.5
Maximum share of cases per variant●	69%	5%	9%	3%	3%	10.5%
Maximum share of cases per variant●●	100%	69%	67%	52%	37%	36%
Mean share of manual activated tasks●	100%	100%	100%	100%	86%	44%
Mean share of manual activated tasks●●	0%	0%	0%	0%	5%	2%

*with at least read access ●mean workplan stage/s ●●mean none workplan stages

Table 6.9.: Case execution behavior summary across sites.

6.7. Summary of Experience from Practice

Hereinafter, we summarize the practical experience gained from the case studies in the context of an international integrated care project (cf. Chapter 6). In total, six case studies evaluated our ACM4IC approach at three different hospital sites, namely Groningen, Tel Aviv, and Lleida located in the Netherlands, Israel, and Spain (cf. Section 6.2). Our approach is purely model-based integrated into the project's architecture and orchestrates the integrated care services (cf. Section 6.3). All services are deployed on a productive cloud environment on AWS to conduct the case studies (cf. Section 6.4). During the case studies, case templates were modeled and evolutionarily improved within approximately 20 months (cf. Section 6.5). The case templates

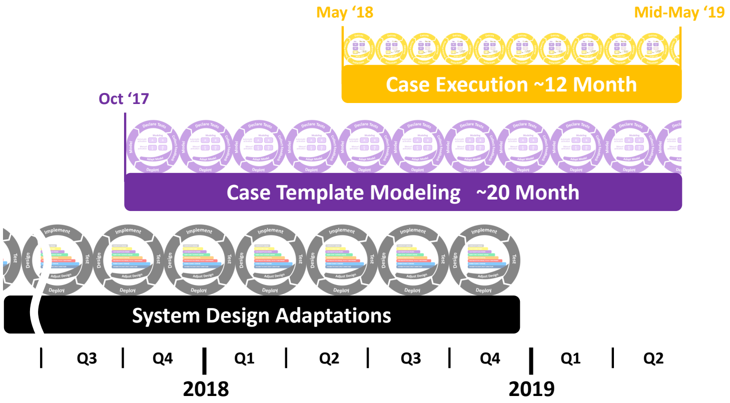


Figure 6.20.: Evaluation timeline.

were instantiated and flexibly adapted during run-time to accomplish patient treatments within approximately one year (cf. Section 6.6). Figure 6.20 illustrates the abstract evaluation levels and time context (cf. Section 6.1). While the case studies continued until the end of the year 2019, the evaluation contains the data collected until mid-May 2019. In the following, we provide a high-level summary structured according to the abstract evaluation levels:

System Design Adaptions Previously, the evaluation did not focus on the system design adaptations. While the conceptual design (cf. Chapter 4) and prototypical implementation (cf. Chapter 5) describe the resulting artifact, this paragraph provides a brief chronological summary. In October 2017, the ACM4IC prototype reached a maturity that enabled declaring case templates with limited functionality. During the case-template modeling and case execution, we identified shortcomings that led to improvements in the conceptual design and in the prototypical implementation accordingly. Concepts such as `DualTasks`, `CustomDataRepresentations`, and the correction of completed or terminated tasks were introduced. Since the beginning of the case studies in May 2018, feedback from daily use was incorporated and led to multiple feature improvements. Actions such as completing a stage, terminating a stage, or deleting a case were exclusively restricted to the case owner. Task `Alerts`, which are also named notifications according to the frontend terminology, were initially only shown on the dashboard of the related task owner. However, the case owner was not actively notified, which was then introduced to address practical needs. With increasing maturity, the number of system design adaptions decreased, and changes were primarily applied to the case templates. While the last change affecting the ACM4IC engine was committed on the 18th of December 2018, the case studies continued, thus indicating that the prototypical implementation reached a maturity that is applicable in such an integrated care context. However, we are aware that our ACM4IC is still a prototypical implementation.

Case Template Modeling The purely meta-model-based ACM4IC approach provides a rich set of capabilities, whereas the usability in practice heavily depends on the quality of the modeled case templates (cf. Section 6.5). A typical case template contains a comparatively high share of human tasks representing medical questionnaires which are relatively simple to model. A medical questionnaire primarily contains multiple questions with different answer options and a dynamical expression-based calculated score. Often threshold-based coloring is applied to simplify the visual score interpretation. Completely customized representations such as the generation of a dynamic SVG body representation based on parameters from a medical questionnaire is costly. However, ACM4IC allows to quickly declare common questionnaires and enables customization where needed. The semantic integration and orchestration of third-party systems is achieved with `DualTasks` each representing a corresponding micro-service. In combination with hooks, an event-based synchronization based on state changes is realized. `SummarySections` are used to represent the treatment-specific patient status for care professionals intuitively and visually attractive. Derived from experience, we summarized case modeling best practice principles (cf. Section 5.3). In total, 44 case template versions were deployed on the production environment within approximately 20 months, averaging approximately 7.3 case template versions per case study.

Case Execution The high-level case execution metrics indicate the typical characteristics of knowledge-intensive processes (cf. Section 6.6.6). Case templates contain multiple stages with a different degree of structure. Stages requiring many context-dependent decisions typically apply flexible process adaptions, which are also named run-time planning. During the workplan stage execution, an individual patient-centric treatment plan is created based on predefined fragments. We analyzed the process execution variants per stage. Comparing the maximum share of process execution variants between non-workplan stages and the workplan stage indicate a significantly lower share for workplan stages. Team roles, messaging, and alerts, which are also named notifications according to the frontend terminology, are used to coordinate the patient-centric treatment. In total, 232 patients across six case studies were treated using the ACM4IC approach. While our evaluation period ended after approximately one year in mid-May, the patient admission was continued, thus the number of patients was still increasing until the end of the integrated care project in December 2019.

Considering all aspects such as the maturity of the case templates and the case execution behavior, the case studies in Lleida are the most mature ones. In Lleida, many small iterations with responsible care professionals led to mature case templates. In daily practice, small case template adaptations significantly increased the usability. Typically, small iterations also lead to an increasing implicit understanding of care professionals regarding what is customizable within the case template. During the case studies in Lleida, 86 engaged care professionals (CONNECARE Consortium, 2019a), treated 75 patients. The clinical leader in Lleida stated:

> *"I met our territorial healthcare manager, he is impressed about CONNECARE, he is willing to scale it up throughout our chronic care network"*
>
> *G. T. in CONNECARE Consortium (2019b, p. 17)*

The applicability of the prototypical ACM4IC implementation is approved in a real-world environment beyond a controlled laboratory experiment across three hospital sites. Occurring site-specific requirements could be expressed within the case templates and the site-specific workspaces. However, for long-term productive usage, multiple adaptations and extensions would be required, such as a visual modeling environment.

CHAPTER 7

Conclusion and Outlook

This chapter recapitulates the thesis and its main objective is to conclude on the contribution of this work. In Section 7.1, the thesis is briefly summarized and the research questions (cf. Section 1.2) are answered. In Section 7.2, a critical reflection and known limitations are presented considering the ACM4IC prototype and the related evaluation with case studies. Finally, Section 7.3 elaborates further research opportunities.

7.1. Summary

This thesis starts with the problem description in Section 1.1. The demographic change in Europe leads to an aging population. Multiple chronical diseases occur frequently above the age of 65 years. Elderly patients with multiple chronic diseases typically consume a disproportionately high share of healthcare resources. Uncoordinated, simultaneous treatments comprise a high uncertainty regarding the benefits and harm caused by side effects. Integrated care is a promising holistic, patient-centered approach to improve the treatment of those patients. The main challenging aspects are the highly context-dependent unpredictable treatments, the lack of semantic information exchange and system interoperability, and the coordination across multiple organizations and different roles. While the need for integrated care is wildly acknowledged, adequate support for integrated care is still missing. Considering the continuously evolving hospital- and treatment-specific requirements, traditional custom implementations or adaptations are inadequate. Based on the problem description, the research questions are formulated in Section 1.2, and the applied research design is presented in Section 1.3. Furthermore, the thesis contribution is summarized in Section 1.4, and a high-level outline is presented in Section 1.5.

In Chapter 2, the foundation and related work are elaborated. An ACM introduction and definition in conjunction with healthcare context is presented in Section 2.1.1. Since CMMN is widely acknowledged as a notation for ACM, practically relevant notation elements are summarized in Section 2.1.2. The classification of existing tools indicates that those provide only partly support relevant capabilities for an ACM4IC approach, as elaborated in Section 2.1.3 and initially published in Michel et al. (2018). Important related work is presented chronologically ordered in Section 2.2.1. The ACM4IC approach and prototype build on previously existing concepts and tools are further described with more details. Hybrid Wiki, an approach enabling dynamically structuring wiki pages with key-value pairs named attributes, whereas a data-first and schema-second strategy enables late data schemata modeling, are presented in Section 2.2.2. Organic Data Science, an approach to collaboratively ah-hoc decompose scientific questions into manageable tasks, is introduced in Section 2.2.3. Darwin incorporates successful concepts from the Organic Data Science and aims to support users in collaboratively structuring knowledge-intensive processes as presented in Section 2.2.4. End-User Analytics, an approach that extends the Hybrid Wiki with a functional query language named Model-Based Expression Language (MxL), is presented in Section 2.2.5. The ACM4IC approach technically builds on the Hybrid Wiki concept, including end-user analytic capabilities while inspired by the technical independent Darwin approach.

After summarizing the first two chapters containing the motivation and introduction as well as the foundation and related work, we answer the research hypothesis raised in Section 1.2:

Research Hypothesis: Adaptive Case Management for Integrated Care (ACM4IC) will empower care professionals with a collaborative, purely meta-model-based software solution customizable to hospital-, treatment-, and patient-specific needs, to enable patient-centric treatments across organizational boundaries.

The research hypothesis was subdivided into specific research questions to structure the overall objective which are answered subsequently. The first research question addresses the generic requirements also reusable to implement related software approaches:

RQ1: What are the key requirements for ACM4IC?

The problem description (cf. Section 1.1) emphasizes three major integrated care challenges that are relevant the ACM4IC approach, namely highly context-dependent unpredictable treatments, semantic information exchange and system interoperability, and coordination across multiple organizations and different roles. In Section 3.1, the derived high-level requirements from the literature are presented. Firstly, to support a purely meta-model-based approach which supporting data schemata models, adaptive process models, role-based and discretionary access right models, and simple user interface models to allow customization where required. Secondly, to support third-party system integration which includes supporting external user identity management, process orchestration of third-party systems, and semantic integration of external data sources. Thirdly, support communication and coordination which includes supporting process-contextual notifications, direct case-based communication, unstructured case notes, case template specific summaries and clarify needed contribution. Accordingly, a summary of the practical challenges and the resulting requirements are provided by Michel and Matthes (2018). The derived requirements are helpful for the conceptual design, which leads to the following research question:

RQ2: What are the key aspects of the ACM4IC conceptual design?

The high-level answer to this research question is provided in Section 4.1, where all required conceptual layers are described and a capability overview is provided. The first layer is the annotated versioned linked content graph, the second layer is the multiple dynamic schemata, the third layer holds the role-based and discretionary access control models, and the fourth layer with the advanced search and indexing result from the Hybrid Wiki meta-model (cf. Section 2.2.2). The fifth layer integrates a higher-order functional language and results from the end-user analytic meta-model extension (cf. Section 2.2.5). The sixth layer holds the case-based process models, the seventh layer the case execution engine, and the eight layer with the simple user interface models are newly created for ACM4IC. The actual resulting unified meta-model, including a conceptual description for each element, is presented in Section 4.2. In addition, a more detailed meta-model containing the major attributes is presented in Section A.1. The strict separation between case definition and data schemata as well as cases and data enables the expression of complex case templates while reusing previously entered data within the following processes to either show the data read-only, modify the data or use the data as meta-data to control the case flow where required. The resulting execution semantics enabling knowledge-intensive processes is elaborated in Section 4.3. The generic lifecycle, including possible state transitions, primary execution semantics, and special cases are also elaborated. During the conceptual design, multiple challenges occurred such as the linkage between process and data layer, ensuring interoperability with non-model-based systems, dealing with human input errors, and the balance between generic reusability and customizability as described in Section 4.4. Furthermore, supported CMMN elements and extensions, including their decorator applicability are presented in Section 4.5. Finally, a matrix summarizes which conceptual layer supports which requirement in Section 4.6. The initial results concerning the meta-model have been published by Hernandez-Mendez et al. (2018) and Michel and Matthes (2018). However, care professionals as end-users require a valuable tool considering those concepts, which leads to the following research question:

RQ3: What are the key aspects of the ACM4IC prototypical implementation?

Resulting from multiple iterative evaluations with care professionals, the end-user interface features are elaborated based on screens showing sample models in Section 5.1. Relevant essential end-user interface features are: a single sign-on and multi-tenancy support in Section 5.1.1, a dashboard to provide a user-specific overview to identify needed contributions quickly in Section 5.1.2, the my-cases view providing a personal list of accessible cases in Section 5.1.3, and the detailed case representation providing case-specific meta-data and enabling navigation to multiple subviews in Section 5.1.4. The detailed case representation feature is further decomposed into a case summary in Section 5.1.5, a case workflow overview enabling a flexible process adaptation in Section 5.1.6, a detailed task in Section 5.1.8, a custom data representation enabling the overriding of default representations in Section 5.1.9, case data in Section 5.1.10, a case team in Section 5.1.11, case notifications in Section 5.1.12, and the case messages feature in Section 5.1.13. Additionally, the initial set end-user of interface features has been summarized by Michel and Matthes (2018). A meta-model-based user interface becomes valuable with the corresponding model. Merely API-based approaches are powerful to declare single meta-model elements. However, they are practically inadequate to declare a comprehensive case template containing strongly interrelated elements. Therefore, case templates are declarable within a single, mostly self-contained XML file. Corresponding to the meta-model elements (cf. Section 4.2), the XML reference elaborates the grammar usable to express comprehensive case templates, including simple samples in Section 5.2. In addition to the actual case template declaration grammar, the optional test case execution grammar enables the creation of reproducible tests to prevent unnecessary run-time errors. Best practice modeling principles are summarized from experience with multiple case studies in Section 5.3. The model import flow elaborates the transformation from an XML case template to an executable case template, including the dependency resolving in Section 5.4. Executable cases must be visually accessible for end-users and technically accessible for the frontend application and external systems. Therefore, the conceptual API design is described in Section 5.5. Several fundamental technical challenges are illustrated in Section 5.6. Prototypical supported requirements are summarized in Section 5.7. Finally, the last research question leads to practical applicability outside of a controlled laboratory environment:

RQ4: What are the experiences using ACM4IC in practice?

In the context of a European Horizon 2020 integrated care project, the ACM4IC approach is applied in a real-world environment beyond a controlled laboratory environment. The Personalised Connected Care for Complex Chronic Patients (CONNECARE) project, including accomplished case studies, is introduced in Section 6.2. Six case studies distributed across three hospital sites located in Groningen, Tel Aviv, and Lleida are described in detail to provide context for our evaluation. The project's conceptual system architecture is presented to demonstrate the applicability of our ACM4IC prototypical implementation within an integrated care environment in Section 6.3. Subsequently, the logical system deployment is described in Section 6.4. The case study modeling process and related artifacts are illustrated in Section 6.5. While case templates across case studies indicate many similarities, each case template is customized to case study specific requirements and represents a unique artifact. Within approximately 20 months, 44 case templates were modeled and deployed on the production environment. Findings regarding the case template modeling process have been published in Michel et al. (2019). The case study execution behavior is analyzed in Section 6.6 accordingly. Within six case studies distributed in three hospital sites, 232 patients were treated within approximately one year. Additionally, the results are summarized on a high abstraction level in Section 6.7. Lleida's aim to use the ACM4IC approach beyond the project scope shows the practical relevance and highlights the impact of this thesis.

A critical reflection and known limitations considering the ACM4IC prototype and the related evaluation with case studies are presented in Section 7.2. Finally, further research opportunities are elaborated in Section 7.3.

7.2. Critical Reflection and Known Limitations

While this thesis is compliant with the presented requirements (cf. Chapter 3), we are aware of certain limitations. The functionally of the prototype is critically recaptured in Section 7.2.1 and the evaluation in Section 7.2.2 accordingly.

7.2.1. Critical Reflection on the Functionality of the Prototype

During the conceptual design and prototypical implementation, design decisions led to conceptual and technical constraints, respectively. This section critically reflects the functionalities according to the conceptual layers, beginning with the lowest, the data layer.

The original Hybrid Wiki model supports late data modeling to enable a data-first, schema-second strategy (cf. Section 2.2.2). The extension layer with end-user analytical capabilities, which realizes a higher-order functional language (cf. Section 2.2.5) requires an explicit data model and expects compliant data (Reschenhofer, 2017, p. 161). Similarly, our approach requires an explicit data schema to enable the declaration of corresponding case templates. The data of an instantiated case depends on the model, typically represented with a hierarchically linked data structure (cf. Section 4.4.1). The required flexibility is reflected through data types and multiplicities, which is expressive considering hierarchically linked structures.

In light of the case template modeling capabilities, the CMMN specification is used as a reference while not fully implemented. The supported modeling elements and applicable decorators are described (cf. Section 4.5). Nested case templates as a subordinate process are not supported. During the case studies within our evaluation, nested case templates were never required but likely useful for very long-running superordinate cases. Nested stages are conceptually considered but never implemented in the frontend and therefore not tested in practice. Sentries allow declaring temporal dependencies between tasks. However, the introduction of parallel repetitions prevents the meaningful evaluation of sentries across processes on different nesting levels which is therefore not supported. For usability reasons, the run-time planning is simplified to allow dynamically adding predefined processes with one operation without a CMMN planning table. However, to support a purely model-based integrated care approach, we introduced additional modeling elements. The dual-task definitions represent a human task definition followed by an automated task definition which is introduced to enable healthcare services interoperability with a single model element (cf. Section 4.4.5). Similarly, a hook definition concept is introduced to enable a purely and flexible model-based integration strategy (cf. Section 4.4.5). According to the CMMN specification (Object Management Group, 2016, p. 113), only task and stages are re-activated when failed or resumed when suspended, but they do not allow the modification of completed or terminated tasks. During the case studies, we noticed that already completed or terminated processes must be correctable to ensure decent usability (cf. Section 4.3.1). Theoretical accompanied consistency and accountability, occurring while correcting completed or terminated processes are challenging (cf. Section 4.4.7). In practice, during the case studies, we learned those challenges are handleable considering that knowledge workers know best what they do, and the system documents those correctional changes visibly on each task element.

Limitation regarding the simple user interface models primary refer to the design simplicity. While the task declarations allow specifying simple custom layouts with a position flag, the layout declaration does not allow to distinguish between an editable task view and a read-only task view. Considering the allocated space for task parameters in the edit view does not proportionally grow compared to the read-only view, the alignment between elements across multiple columns might be displaced (cf. Section 4.2 and 5.1.8). I.e., a task parameter representing a string is comparable proportionally whereas an enumeration option shows all options in edit mode and only the selected option in read-only mode. Custom layouts are optimized to either the edit or the read-only view. Default layouts are not affected, all task parameters are listed below each other in one column. With an additional model parameter, read-only and editable layouts could be distinguished.

The case-template modeling process is currently based on an XML declaration that must be imported (cf. Section 5.2). A visual graphical modeling environment is not available. While an experienced case template modeler is able to apply changes quickly on an XML declaration, a visual modeling environment would enable end-users to adapt case templates easily. Case templates support declaring test executions to detect run-time errors quickly after changes. The test execution requires a case template import right before the test execution to create a mapping between the declared XML identifiers, which are unique within the case template and the actual database identifies which are automatically generated and unique across all case templates.

7.2.2. Critical Reflection on the Evaluation

In addition to the presented evaluation (cf. Chapter 6), this section provides a critical reflection to assess the validity of the conclusion. Multiple evaluation strategies are applied to ensure a comprehensive approach assessment, which is critically reflected afterwards.

The evaluation is based on implementation studies of single European integrated care project with four participating hospital sites (cf. Section 6.2). The involved hospital sites performed two to three medical case studies each. Our approach is evaluated on three hospital sites with two case studies each and six case studies in total. Across the sites and case studies, we noticed many similar challenges. However, every additional case study would indicate supplementary challenges and would contribute to ensuring the universal applicability for integrated care. A comparison with the fourth hospital site using a custom implementation based on an ACM engine would be helpful for evaluation purposes. Relevant comparison dimensions are i) the degree of integration with hospital information systems, ii) the communication and coordination capabilities, iii) adaptation capabilities including adaption speed, iv) customizability, and v) potential reusability in other hospital sites. Considering the case study timing, we could not incorporate the potentially valuable comparison into our evaluation strategy. Our evaluation focuses on aspects of the ACM4IC. The expected positive impact by integrated care on patients is evaluated by clinical partners belonging to the project consortium. Hereinafter, the critical reflection is structured according to the evaluation sections.

The described integrated care project aims to provide a SaaS solution. Therefore, our approach has been integrated into the project's architecture and is deployed on the AWS cloud (cf. Section 6.3f.). Moreover, our approach integrates multiple healthcare services, ensuring system interoperability but within a single project context.

For each case study, a case template is modeled, for which the evolution is tracked within a repository (cf. Section 6.5). The iterative case template modeling process is analyzed based on the repository commits, which are classified into three categories. Depending on a modeler, the amount and complexity of changes per commit differ. Even a complex change may be copied from an existing template. A commit represents a conceptually completed change. Weighting commits and similarly approaches are error-prone and should not be applied. However, this fact limits the meaningfulness.

During the case studies, the case templates are instantiated and executed to perform patient treatments. The case execution behavior across different hospital sites and case studies is analyzed (cf. Section 6.6). Considering the low number of instantiated case templates per case study, different case template versions are not explicitly considered. While the organizational setup of the implementation study is close to production, engaging active participation of all roles involved in a case is challenging, which affects the usage of collaboration and coordination features.

7.3. Future Research Opportunities

In the previous sections, we summarized the results and limitations, whereas this section addresses further research opportunities. Inspired by practical needs identified while developing our approach, or based on our iterative PDSA evaluation, we highlight and summarize the three most promising research opportunities.

Visual case template modeling environment During our iterative evaluation (cf. Chapter 6), we identified the need for a visual full-stack case template modeling approach. Modeling practical uses cases as case templates requires domain knowledge, conceptual abstraction capabilities, and technical skills to express modeling elements. Domain knowledge and conceptual abstraction capabilities are prerequisites, but technical skills may be needless with a sophisticated modeling user interface. A primary challenge is potentially managing the complex relations between modeling elements and the related consistency issues within a user-friendly interface.

Analytical capabilities to support modelers and knowledge workers While we investigated slightly into analytical capabilities for evaluation purposes (cf. Section 6.6), those might be interesting for domain experts and modelers to gain insights how the case templates are used and potentially improved (Günther et al., 2006). Traditionally, knowledge-intensive processes are rarely predictable and allow a huge degree of execution freedom. A first step might be to visualize the case execution paths and provide aggregated case template path heat-maps for each case template. In our evaluation context, this would help care professionals track patient pathways visually to identify potential model improvements. In a second step, this is potentially helpful to model clinical decisions. Within a third step, individual case decisions could be supported with recommendations learned from the previous case executions to support knowledge workers.

Examinating applicability in further knowledge-intensive domains Since our underlying concepts are rather generic, examinating the applicability in further domains might be promising, as the multi-domain applicability data-centric core illustrated by Hernandez-Mendez et al. (2018) shows. Applying our approach to an additional knowledge-intensive domain potentially requires small extensions and adaptations of the modeling capabilities (cf. Section 4.2). With each additional domain, the modeling capabilities would emerge and potentially lead to a sophisticated multi-domain adaptive case management modeling approach.

Bibliography

Ritu Agarwal, Guodong (Gordon) Gao, Catherine DesRoches, and Ashish K. Jha. Research Commentary—The Digital Transformation of Healthcare: Current Status and the Road Ahead. *Information Systems Research*, 21(4):796–809, November 2010. doi: 10.1287/isre.1100.0327.

Ian F. Alexander. A taxonomy of stakeholders: Human roles in system development. *International Journal of Technology and Human Interaction (IJTHI)*, 1(1):23–59, January 2005. doi: 10.4018/jthi.2005010102.

Ariel C. Avgar, Prasanna Tambe, and Lorin M. Hitt. Built to learn: How work practices affect employee learning during healthcare information technology implementation. *Management Information Systems Quarterly*, 42(2):645–659, 2018. doi: 10.25300/MISQ/2018/13668.

Thomas Büchner. *Introspektive modellgetriebene Softwareentwicklung*. Dissertation, Technische Universität München, München, Germany, 2007.

David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, September 2014. doi: 10.1109/MCC.2014.51.

Anna Bánáti, Eszter Kail, Krisztian Karóczkai, and Miklos Kozlovszky. Authentication and authorization orchestrator for microservice-based software architectures. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1180–1184, May 2018. doi: 10.23919/MIPRO.2018.8400214.

Simon Bönisch. Discovering clinical pathways of an adaptive integrated care environment. Master's thesis, Technische Universität München, Munich, Germany, 2019.

Nathan Bos, Ann Zimmerman, Judith Olson, Jude Yew, Jason Yerkie, Erik Dahl, and Gary Olson. From shared databases to communities of practice: A taxonomy of collaboratories. *Journal of Computer-Mediated Communication*, 12(2):652–672, January 2007. doi: 10.1111/j.1083-6101.2007.00343.x.

Peter Buneman, Shamim Naqvi, Val Tannen, and Limsson Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, September 1995. doi: 10.1016/0304-3975(95)00024-Q. Fourth International Conference on Database Theory (ICDT '92).

Emily V. Burns. Case management 101: 10 things you must know about case management. In Layna Fischer, editor, *Taming the Unpredictable: Real World Adaptive Case Management: Case Studies and Practical Guidance*, pages 17–25. Future Strategies Incorporated, Florida, USA, 2011. ISBN 978-0-9819870-8-8.

Isaac Cano, Albert Alonso, Carme Hernandez, Felip Burgos, Anael Barberan-Garcia, Jim Roldan, and Josep Roca. An adaptive case management system to support integrated care services: Lessons learned from the NEXES project. *Journal of Biomedical Informatics*, 55: 11–22, 2015. doi: 10.1016/j.jbi.2015.02.011.

Isaac Cano, Ivan Dueñas-Espín, Carme Hernandez, Jordi de Batlle, Jaume Benavent, Juan Carlos Contel, Erik Baltaxe, Joan Escarrabill, Juan Manuel Fernández, Judith Garcia-Aymerich, Miquel Àngel Mas, Felip Miralles, Montserrat Moharra, Jordi Piera, Tomas Salas, Sebastià Santaeugènia, Nestor Soler, Gerard Torres, Eloisa Vargiu, Emili Vela, and Josep Roca. Protocol for regional implementation of community-based collaborative management of complex chronic patients. *npj Primary Care Respiratory Medicine*, 27(1):44, July 2017. doi: 10.1038/s41533-017-0043-9.

Mary E. Charlson, Peter Pompei, Kathy L. Ales, and C.Ronald MacKenzie. A new method of classifying prognostic comorbidity in longitudinal studies: Development and validation. *Journal of Chronic Diseases*, 40(5):373–383, 1987. doi: 10.1016/0021-9681(87)90171-8.

CONNECARE Consortium. Factsheet: Personalized Connected Care for Complex Chronic Patients, May 2016. URL https://www.connecare.eu/wp-content/uploads/2017/04/Project_Factsheet.pdf. Last accessed on April 8, 2019.

CONNECARE Consortium. Project Board Meeting: WP6 Deployment of Clinical Studies Presentation, March 2019a. Modena, Italy. WP leader: Assuta Medical Centers (ASSUTA).

CONNECARE Consortium. Second periodic project review meeting with European Commission reviewers: Overview and Major Achievements presentation, January 2019b. Luxemburg. Leader: Eurecat Technology Center (EURECAT).

CONNECARE Consortium. Deliverable 1.4: Second Periodic Report. European Commission Participant Portal, January 2019c. Luxemburg.

Jordi de Batlle, Eloisa Vargiu, Gerard Torres, Mireia Massip, Felix Michel, Florian Matthes, Felip Miralles, and Ferran Barbé. Implementation of an Integrated Care Platform for the Management of Complex Chronic Patients in Lleida, Spain. In *D37. Topics in Global Health Services Research*, pages A6246–A6246. American Thoracic Society, May 2019. doi: 10.1164/ajrccm-conference.2019.199.1_MeetingAbstracts.A6246.

Jan De Clercq. Single sign-on architectures. In *Infrastructure Security*, pages 40–58, Berlin, Heidelberg, 2002. Springer. ISBN 978-3-540-45831-9.

Henk De Man. Case management: A review of modeling approaches. *BPTrends*, 2009, January 2009. URL https://www.bptrends.com/case-management-a-review-of-modeling-approaches/. Last accessed on January 6, 2019.

William Edwards Deming. *The New Economics for Industry, Government, Education*. The MIT Press. MIT Press, 2018. ISBN 978-0-262-53593-9.

Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. Knowledge-intensive processes: An overview of contemporary approaches. In *Proceedings of the 1st International Workshop on Knowledge-intensive Business Processes (KiBP 2012)*, pages 33–47, Rome, Italy, June 2012.

Julie Doyle, Evert-Jan Hoogerwerf, Janneke Kuiper, Emma Murphy, Caoimhe Hannigan, John Dinsmore, Thomas van der Auwermeulen, Jonas Albert, An Jacobs, Lorenza Maluccelli, Lorenzo Desideri, and Valentina Fiordelmondo. Designing a proactive, person-centred, digital integrated care system. *International Journal of Integrated Care (IJIC)*, 17(5):A211, October 2017. doi: 10.5334/ijic.3521.

Nicolas Ducheneaut and Victoria Bellotti. E-mail As Habitat: An Exploration of Embedded Personal Information Management. *interactions*, 8(5):30–38, September 2001. doi: 10.1145/382899.383305.

European Commission. *Population ageing in Europe: facts, implications and policies*. Publications Office of the European Union, Luxembourg, July 2014. doi: 10.2777/60452.

Jeffrey A. Ferguson and Morris Weinberger. Case management programs in primary care. *Journal of General Internal Medicine*, 13(2):123–126, February 1998. doi: 10.1046/j.1525-1497.1998.00029.x.

Juan Manuel Fernández, Marco Mamei, Stefano Mariani, Miralles Felip, Steblin Alexander, Eloisa Vargiu, and Franco Zambonelli. Towards argumentation-based recommendations for personalised patient empowerment. In *Proceedings of the 2nd International Workshop on Health Recommender Systems*. ACM, August 2017. doi: 10.1145/3109859.3109955.

Robert G Fichman, Rajiv Kohli, and Ranjani Krishnan. The role of information systems in healthcare: Current research and future trends. *Information Systems Research*, 22(3):419–428, September 2011. doi: 10.1287/isre.1110.0382.

Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation, University of California, Irvine, USA, 2000.

Layna Fischer. *Taming the Unpredictable: Real World Adaptive Case Management: Case Studies and Practical Guidance*. Excellence in practice series. Future Strategies Incorporated, Florida, USA, 2011. ISBN 978-0-9819870-8-8.

Sebastian Garde and Petra Knaup. Requirements engineering in health care: the example of chemotherapy planning in paediatric oncology. *Requirements Engineering*, 11(4):265–278, September 2006. doi: 10.1007/s00766-006-0029-6.

Michael Gebhart, Pascal Giessler, and Sebastian Abeck. Restful webservices mit qualität - teil 1: Mit best practices zu einem qualitätsorientierten entwurf. *Objektspektrum*, 2015(1):28–33, 2015a.

Michael Gebhart, Pascal Giessler, and Sebastian Abeck. Restful webservices mit qualität - teil 2: Priorisierung von best practices mittels qualitätsmerkmalen. *Objektspektrum*, 2015(2):58–61, 2015b.

Yolanda Gil, Felix Michel, Varun Ratnakar, and Matheus Hauder. A semantic, task-centered collaborative framework for science. In *The Semantic Web: ESWC 2015 Satellite Events*, pages 58–61, Cham, 2015a. Springer International Publishing. ISBN 978-3-319-25639-9.

Yolanda Gil, Felix Michel, Varun Ratnakar, Matheus Hauder, Christopher Duffy, Hilary Dugan, and Paul Hanson. A task-centered framework for computationally-grounded science collaborations. In *2015 IEEE 11th International Conference on e-Science*, pages 352–361, Munich, Germany, August 2015b. IEEE. doi: 10.1109/eScience.2015.76.

Yolanda Gil, Felix Michel, Varun Ratnakar, Jordan Read, Matheus Hauder, Christopher Duffy, Paul Hanson, and Hilary Dugan. Supporting open collaboration in science through explicit and linked semantic description of processes. In *European Semantic Web Conference*, pages 591–605, Cham, May 2015c. Springer International Publishing. doi: 10.1007/978-3-319-18818-8_36.

Christian W. Günther, Stefanie Rinderle, Manfred Reichert, and Wil van der Aalst. Change mining in adaptive process management systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 309–326, Berlin, Heidelberg, 2006. Springer. doi: 10.1007/11914853_19.

Matheus Hauder. *Empowering End-Users to Collaboratively Structure Knowledge-Intensive Processes*. Dissertation, Technische Universität München, München, Germany, 2016.

Matheus Hauder, Dominik Münch, Felix Michel, Alexej Utz, and Florian Matthes. Examining adaptive case management to support processes for enterprise architecture management. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, pages 23–32. IEEE, September 2014. doi: 10.1109/EDOCW.2014.13.

Matheus Hauder, Simon Pigat, and Florian Matthes. Research challenges in adaptive case management: A literature review. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, pages 98–107, Ulm, Germany, September 2014. doi: 10.1109/EDOCW.2014.24.

Matheus Hauder, Rick Kazman, and Florian Matthes. Empowering end-users to collaboratively structure processes for knowledge work. In *Business Information Systems*, pages 207–219, Cham, 2015. Springer International Publishing. doi: 10.1007/978-3-319-19027-3_17.

Michael Herdman, Claire Gudex, Aandrew Lloyd, M.F. Bas Janssen, Paul Kind, David William Parkin, Gouke J. Bonsel, and Xavier Badia. Development and preliminary testing of the new five-level version of EQ-5D (EQ-5D-5L). *Quality of Life Research*, 20(10):1727–1736, December 2011. doi: 10.1007/s11136-011-9903-x.

Adrian Hernandez-Mendez, Felix Michel, and Florian Matthes. A practice-proven reference architecture for model-based collaborative information systems. *Enterprise Modelling and Information Systems Architectures*, 13(2018):262–273, February 2018. doi: 10.18417/emisa.si.hcm.20.

Wolfgang Hesse and Heinrich C. Mayr. Modellierung in der softwaretechnik: eine bestandsaufnahme. *Informatik-Spektrum*, 31(5):377–393, October 2008. doi: 10.1007/s00287-008-0276-7.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *Management Information Systems Quarterly*, 28(1):75–105, March 2004. ISSN 0276-7783.

Lorin M. Hitt and Prasanna Tambe. Health care information technology, work organization, and nursing home performance. *ILR Review*, 69(4):834–859, March 2016. doi: 10.1177/0019793916640493.

David Hollingsworth. Healthcare. In Keith D. Swenson, editor, *Mastering the unpredictable: how adaptive case management will revolutionize the way that knowledge workers get things done*, chapter 8, pages 163–179. Meghan-Kiffer Press, Tampa, Florida, USA, 2010. ISBN 978-0-929652-12-2.

Jan Horsky, Jiajie Zhang, and Vimla L. Patel. To err is not entirely human: complex technology and user cognition. *Journal of Biomedical Informatics*, 38(4):264–266, 2005. doi: 10.1016/j.jbi.2005.05.002. Special Section: JAMA Commentaries.

Rachelle Kaye, Khaled Abu-Hossien, Felip Miralles, Eloisa Vargiu, and Bella Azaria. From connected care to integrated care–a work in progress. *International Journal of Integrated Care (IJIC)*, 17(5):A448, October 2017. doi: 10.5334/ijic.3768.

Matthias Kurz, Werner Schmidt, Albert Fleischmann, and Matthias Lederer. 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*, S-BPM ONE '15, pages 4:1–4:9, New York, NY, USA, 2015. ACM. doi: 10.1145/2723839.2723843.

Yu-Kai Lin, Hsinchun Chen, Randall A. Brown, Shu-Hsing Li, and Hung-Jen Yang. Healthcare Predictive Analytics for Risk Profiling in Chronic Care: A Bayesian Multitask Learning Approach. *Management Information Systems Quarterly*, 41(2):473–495, 2017. doi: 10.25300/MISQ/2017/41.2.07.

Stefano Mariani, Eloisa Vargiu, Marco Mamei, Franco Zambonelli, and Felip Miralles. Deliver intelligence to integrate care: the Connecare way. *International Journal of Integrated Care (IJIC)*, 19(4):A176, August 2019. doi: 10.5334/ijic.s3176.

Mike A. Marin, Matheus Hauder, and Florian Matthes. Case management: An evaluation of existing approaches for knowledge-intensive processes. In *Business Process Management Workshops*, pages 5–16, Cham, 2016. Springer International Publishing. doi: 10.1007/978-3-319-42887-1_1.

Florian Matthes, Christian Neubert, and Alexander Steinhoff. Hybrid wikis: Empowering users to collaboratively structure information. In *Proceedings of 6th International Conference on Software and Data Technologies (ICSOFT)*, pages 250–259, 2011.

John T. Matthias. Technology for case management. In Keith D. Swenson, editor, *Mastering the unpredictable: how adaptive case management will revolutionize the way that knowledge workers get things done*, chapter 4, pages 63–88. Meghan-Kiffer Press, Tampa, Florida, USA, 2010. ISBN 978-0-929652-12-2.

John T. Matthias. User requirements for a new generation of case management systems. In Layna Fischer, editor, *Taming the Unpredictable: Real World Adaptive Case Management: Case Studies and Practical Guidance*, pages 45–52. Future Strategies Incorporated, Florida, USA, 2011. ISBN 978-0-9819870-8-8.

Deborah J. Mayhew. *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1st edition, 1999. ISBN 9-781558-605619.

Dermont McCauley. Acm and business agility for the microsoft-aligned organization. In Layna Fischer, editor, *Taming the Unpredictable: Real World Adaptive Case Management: Case Studies and Practical Guidance*, pages 65–75. Future Strategies Incorporated, Florida, USA, 2011. ISBN 978-0-9819870-8-8.

Erik Meijer, Brian Beckman, and Gavin Bierman. Linq: Reconciling object, relations and xml in the .net framework. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 706–706, New York, NY, USA, June 2006. ACM. doi: 10.1145/1142473.1142552.

Felix Michel. A structured task-centered framework for online collaboration. Master's thesis, Technische Universität München, Munich, Germany, 2014.

Felix Michel and Florian Matthes. A holistic model-based adaptive case management approach for healthcare. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, 6th International Workshop on Adaptive Case Management and other non-workflow approaches to BPM, pages 17–26, Stockholm, Sweden, October 2018. IEEE. doi: 10.1109/EDOCW.2018.00014.

Felix Michel, Yolanda Gil, Varun Ratnakar, and Matheus Hauder. A virtual crowdsourcing community for open collaboration in science processes. In *21st Americas Conference on Information Systems (AMCIS)*, Fajardo, Puerto Rico, August 2015a.

Felix Michel, Yolanda Gil, Varun Ratnakar, and Matheus Hauder. A Task-Centered Interface for On-Line Collaboration in Science. In *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*, pages 45–48, Atlanta, Georgia, USA, March 2015b. ACM. doi: 10.1145/2732158.2732181.

Felix Michel, Adrian Hernandez-Mendez, and Florian Matthes. An overview of tools for an integrated and adaptive healthcare approach. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, 6th International Workshop on Adaptive Case Management and other non-workflow approaches to BPM, pages 27–32, Stockholm, Sweden, October 2018. IEEE. doi: 10.1109/EDOCW.2018.00015.

Felix Michel, Sven-Volker Rehm, and Florian Matthes. Keep up with care: Researching system adaptability in chronic care management of elderly patients. In *Proceedings of the 27th European Conference on Information Systems (ECIS)*, Stockholm and Uppsala, Sweden, June 2019. ISBN 978-1-7336325-0-8.

Hamid R. Motahari-Nezhad and Keith D. Swenson. Adaptive case management: Overview and research challenges. In *2013 IEEE 15th Conference on Business Informatics*, pages 264–269. IEEE, July 2013. doi: 10.1109/CBI.2013.44.

Christian Neubert. *Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms*. Dissertation, Technische Universität München, München, Germany, 2012.

Object Management Group. Business Process Model And Notation (BPMN), March 2007. URL https://www.omg.org/spec/BPMN/1.0/PDF. Last accessed on May 12, 2019.

Object Management Group. Business Process Model And Notation (BPMN), January 2014a. URL https://www.omg.org/spec/BPMN/2.0.2/PDF. Last accessed on May 12, 2019.

Object Management Group. Case Management Model and Notation (CMMN), January 2014b. URL https://www.omg.org/spec/CMMN/1.0/PDF. Last accessed on July 5, 2019.

Object Management Group. Object Constraint Language (OCL), February 2014c. URL https://www.omg.org/spec/OCL/2.4/PDF. Last accessed on January 19, 2019.

Object Management Group. Unified Modeling Language (UML), March 2015. URL https://www.omg.org/spec/UML/2.5/PDF. Last accessed on January 19, 2019.

Object Management Group. Case Management Model and Notation (CMMN), January 2016. URL https://www.omg.org/spec/CMMN/1.1/PDF. Last accessed on July 5, 2019.

Peter Osvath, Viktor Voros, A. Kovacs, Ildiko Greges, S. Fekete, T. Tenyi, and S. Fekete. The ict4life project–design and development of a new information technology platform for patients with alzheimer's disease. *European Psychiatry*, 41:545–546, 2017. doi: 10.1016/j.eurpsy.2017.01.765. Abstract of the 25th European Congress of Psychiatry.

Marielle Ouwens, Hub Wollersheim, Rosella Hermens, Marlies Hulscher, and Richard Grol. Integrated care programmes for chronically ill patients: a review of systematic reviews. *International Journal for Quality in Health Care*, 17(2):141–146, January 2005. doi: 10.1093/intqhc/mzi016.

Jens Rasmussen and Kim J. Vicente. Coping with human errors through system design: implications for ecological interface design. *International Journal of Man-Machine Studies*, 31(5): 517–534, November 1989. doi: 10.1016/0020-7373(89)90014-X.

Manfred Reichert and Hajo A. Reijers, editors. *Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers*, volume 256 of *Lecture Notes in Business Information Processing*. Springer, June 2016. doi: 10.1007/978-3-319-42887-1.

Thomas Reschenhofer. Design and prototypical implementation of a model-based structure for the definition and calculation of enterprise architecture key performance indicators. Master's thesis, Technische Universität München, Munich, Germany, 2013.

Thomas Reschenhofer. *Empowering End-users to Collaboratively Analyze Evolving Complex Linked Data*. Dissertation, Technische Universität München, München, Germany, 2017.

Thomas Reschenhofer and Florian Matthes. 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*, pages 96–100, 2016a. doi: 10.1109/VLHCC.2016.7739670.

Thomas Reschenhofer and Florian Matthes. Empowering end-users to collaboratively manage and analyze evolving data models. In *Twenty-second Americas Conference on Information Systems (AMCIS)*, San Diego, 2016b.

Thomas Reschenhofer, Manoj Bhat, Adrian Hernandez-Mendez, and Florian Matthes. Lessons learned in aligning data and model evolution in collaborative information systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 132–141, New York, NY, USA, 2016. IEEE. doi: 10.1145/2889160.2889235.

Robert Koch-Institute. *Health in Germany*. Federal Health Reporting. Jointly provided by RKI and Destatis, Robert Koch Institute, Nordufer 20, 13353 Berlin, Germany, 2015. doi: 10.17886/rkipubl-2015-010.

Robert Koch-Institute. *Health in Germany – the most important developments*. Federal Health Reporting. Jointly provided by RKI and Destatis, Robert Koch Institute, Nordufer 20, 13353 Berlin, Germany, 2016. doi: 10.17886/RKI-GBE-2017-036.

Vassilis Solachidis, Ioannis Paliokas, Nicholas Vretos, Konstantinos Votis, Ulises Cortés, and Dimitrios Tzovaras. Two examples of online ehealth platforms for supporting people living with cognitive impairments and their caregivers. In *Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference*, PETRA '18, pages 449–454, New York, NY, USA, 2018. ACM. doi: 10.1145/3197768.3201556.

Diane M. Strong and Steven M. Miller. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems (TOIS)*, 13(2):206–233, April 1995. doi: 10.1145/201040.201049.

Ana-Maria Suduc, Mihai BÎuoi, and Florin Gheorghe Filip. User awareness about information systems usability. *Studies in Informatics and Control*, 19(2), June 2010. doi: 10.24846/v19i2y201004.

Keith D. Swenson. *Mastering the unpredictable: how adaptive case management will revolutionize the way that knowledge workers get things done*. Meghan-Kiffer Press, Tampa, Florida, USA, 2010. ISBN 978-0-929652-12-2.

Keith D. Swenson. Case management: contrasting production vs. adaptive. In Layna Fischer, editor, *How knowledge workers get things done: Real-world adaptive case management*, Excellence in practice series, pages 109–116. Future Strategies, 2012. ISBN 978-0-9849764-4-7.

Mary E. Tinetti, Terri R. Fried, and Cynthia M. Boyd. Designing Health Care for the Most Common Chronic Condition—MultimorbidityMultimorbidity Care—A Common Chronic Condition. *JAMA*, 307(23):2493–2494, June 2012. doi: 10.1001/jama.2012.5265.

Noam Tractinsky, Adi S. Katz, and Dror Ikar. What is beautiful is usable. *Interacting with Computers*, 13(2):127–145, December 2000. doi: 10.1016/S0953-5438(00)00031-X.

Alexandros T. Tzallas, Nikolaos Katertsidis, Konstantinos Glykos, Sofia Segkouli, Konstantinos Votis, Dimitrios Tzovaras, Cristian Barrué, Ioannis Paliokas, and Ulises Cortés. Designing a gamified social platform for people living with dementia and their live-in family caregivers. In *Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference*, PETRA '18, pages 476–481, New York, NY, USA, 2018. ACM. doi: 10.1145/3197768.3201560.

Pim P. Valentijn, Sanneke M. Schepman, Wilfrid Opheij, and Marc A. Bruijnzeels. Understanding integrated care: a comprehensive conceptual framework based on the integrative functions of primary care. *International Journal of Integrated Care (IJIC)*, 13(1), March 2013. doi: 10.5334/ijic.886.

Wil M. P. van der Aalst and Paul J. S. Berens. Beyond workflow management: Product-driven case handling. In *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '01, pages 42–51, New York, NY, USA, 2001. ACM. doi: 10.1145/500286.500296.

Wil M.P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162, 2005. doi: 10.1016/j.datak.2004.07.003.

Eloisa Vargiu, Juan Manuel Fernàndez, Felip Miralles, Isaac Cano, Elena Gimeno-Santos, Carme Hernandez, Gerard Torres, Jordi Colomina, Jordi de Batlle, Rachelle Kaye, Bella Azaria, Shauli Nakar, M.H. Lahr, Esther Metting, Margot Jager, Hille Meetsma, Stefano Mariani, Marco Mamei, Franco Zambonelli, Felix Michel, Florian Matthes, Jo Goulden, John Eaglesham, and Charles Lowe. Integrated Care for Complex Chronic Patients. *International Journal of Integrated Care (IJIC)*, 17(5):A302, October 2017. doi: 10.5334/ijic.3619.

Eloisa Vargiu, Juan Manuel Fernández, Mauricio Gonzales-Gonzales, Juan Manuel Morales-Garzón, Kitiara Prunera-Moreda, and Felip Miralles. Self-management of complex chronic patients: Needs and a proposal. In *Proceedings of the Fourth Italian Workshop on Artificial Intelligence for Ambient Assisted Living (AI*AAL.it 2018)*, volume 2333, pages 37–50, November 2018a.

Eloisa Vargiu, Juan Manuel Fernández, and Felip Miralles. Patient empowerment in connecare. *International Journal of Integrated Care (IJIC)*, 18(S2):258, October 2018b. doi: 10.5334/ijic.s2258.

Eloisa Vargiu, Juan Manuel Fernández, Mauricio Gonzales-Gonzales, Juan Manuel Morales-Garzón, Kitiara Prunera-Moreda, and Felip Miralles. A self-management system for complex chronic patients. *International Journal of Integrated Care (IJIC)*, 19(S1):A101, August 2019a. doi: 10.5334/ijic.s3101.

Eloisa Vargiu, Gerard Torres, Mireia Massip, Juan Manuel Fernández, Felix Michel, Florian Matthes, and Felip Miralles. Connected care for complex chronic patients in lleida. *International Journal of Integrated Care (IJIC)*, 19(4), August 2019b. doi: 10.5334/ijic.s3102.

Carlos A. Velasco, Yehya Mohamad, and Philip Ackermann. Architecture of a web of things ehealth framework for the support of users with chronic diseases. In *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*, DSAI 2016, pages 47–53, New York, NY, USA, 2016. ACM. doi: 10.1145/3019943.3019951.

Michael White. Case management: Combining knowledge with process. *BPTrends*, July 2009. URL https://www.bptrends.com/case-management-combining-knowledge-with-process/. Last accessed on May 6, 2019.

World Health Organization. *Integrated care models: an overview.* WHO Regional Office for Europe, UN City, Marmorvej 51, DK-2100 Copenhagen Ø, Denmark, 2016. URL http://www.euro.who.int/__data/assets/pdf_file/0005/322475/Integrated-care-models-overview.pdf.

Abbreviations

ACM Adaptive Case Management

ACM4IC Adaptive Case Management for Integrated Care

API Application Programming Interface

AWS Amazon Web Services

BPM Business process management

BPMN Business Process Model and Notation

CCP Complex Chronic Patients

CMMN Case Management Model and Notation

CONNECARE Personalised Connected Care for Complex Chronic Patients

COPD Chronic Obstructive Pulmonary Disease

CS1 Case Study 1

CS2 Case Study 2

EC2 Elastic Compute Cloud

gzip GNU zip

HIS Hospital Information System

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IS Information System

ISR Information Systems Research

IT Information Technology

JDBC Java Database Connectivity

JSON JavaScript Object Notation

JWT JSON Web Token

KPI Key Performance Indicator

LTR Left To Right

MxL Model-Based Expression Language

OCL Object Constraint Language

ODS Organic Data Science

ORM Object-Relational Mapping

PCM Production Case Management

PDSA Plan Do Study Act

REST Representational State Transfer

RTL Right To Left

SaaS Software as a Service

SACM Smart Adaptive Case Management

SAX Simple API for XML

SMS Self-Management System

SPA Single Page Application

SQL Structured Query Language

SSO Single Sign-On

SVG Scalable Vector Graphics

UI User Interface

UIM User Identity Management

UML Unified Modeling Language

URL Uniform Resource Locator

UTF-8 8-Bit Universal Coded Character Set Transformation Format

WP Work Package

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

APPENDIX A

Appendix

A.1. Detailed Meta-Model

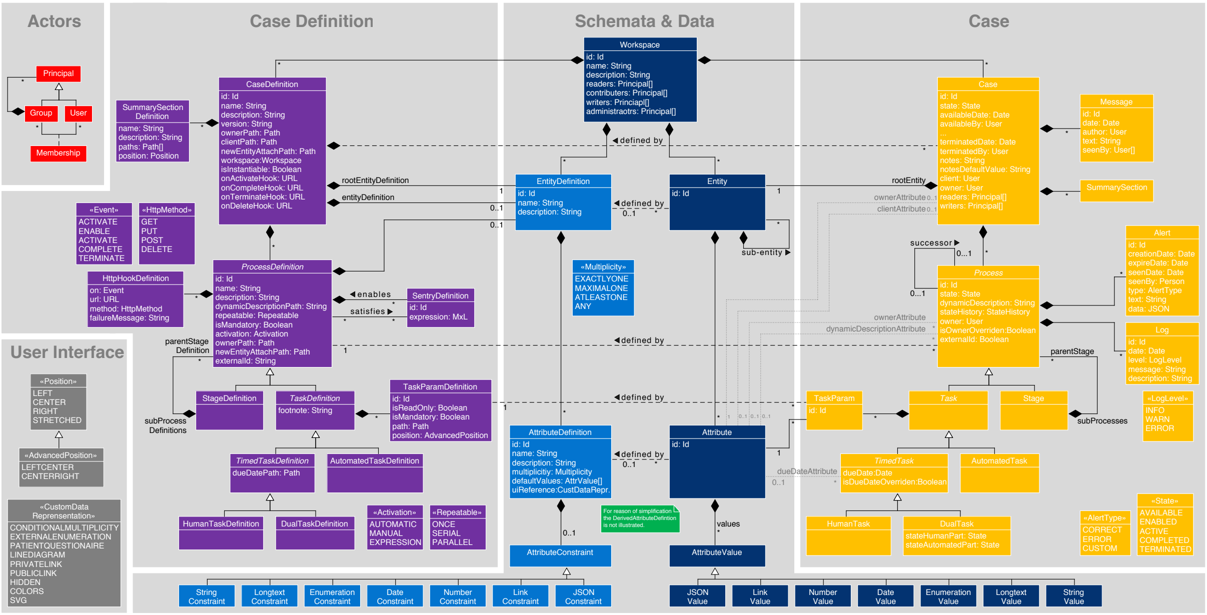


Figure A.1.: Detailed meta-model.

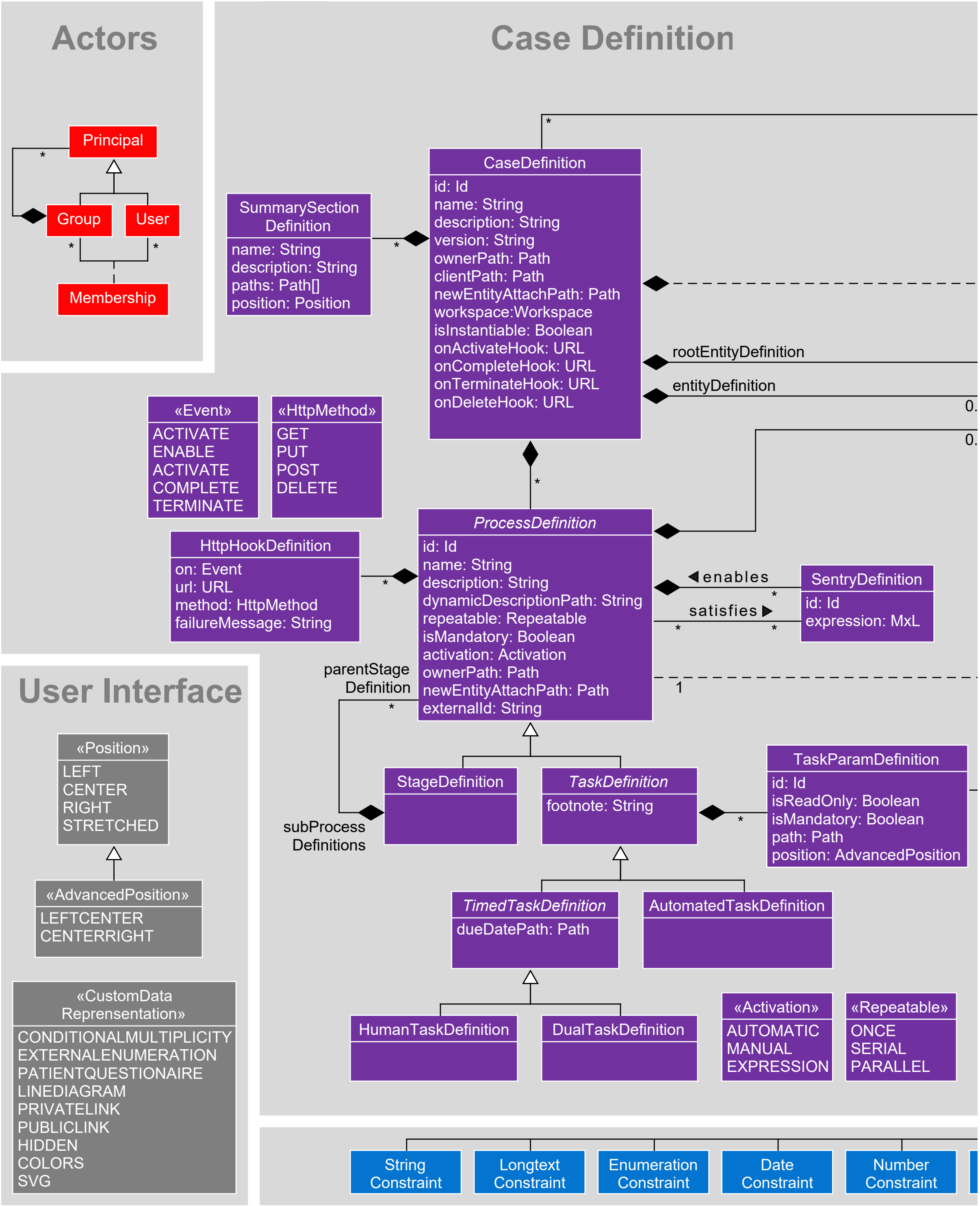


Figure A.2.: Detailed meta-model with focus on the case definition.

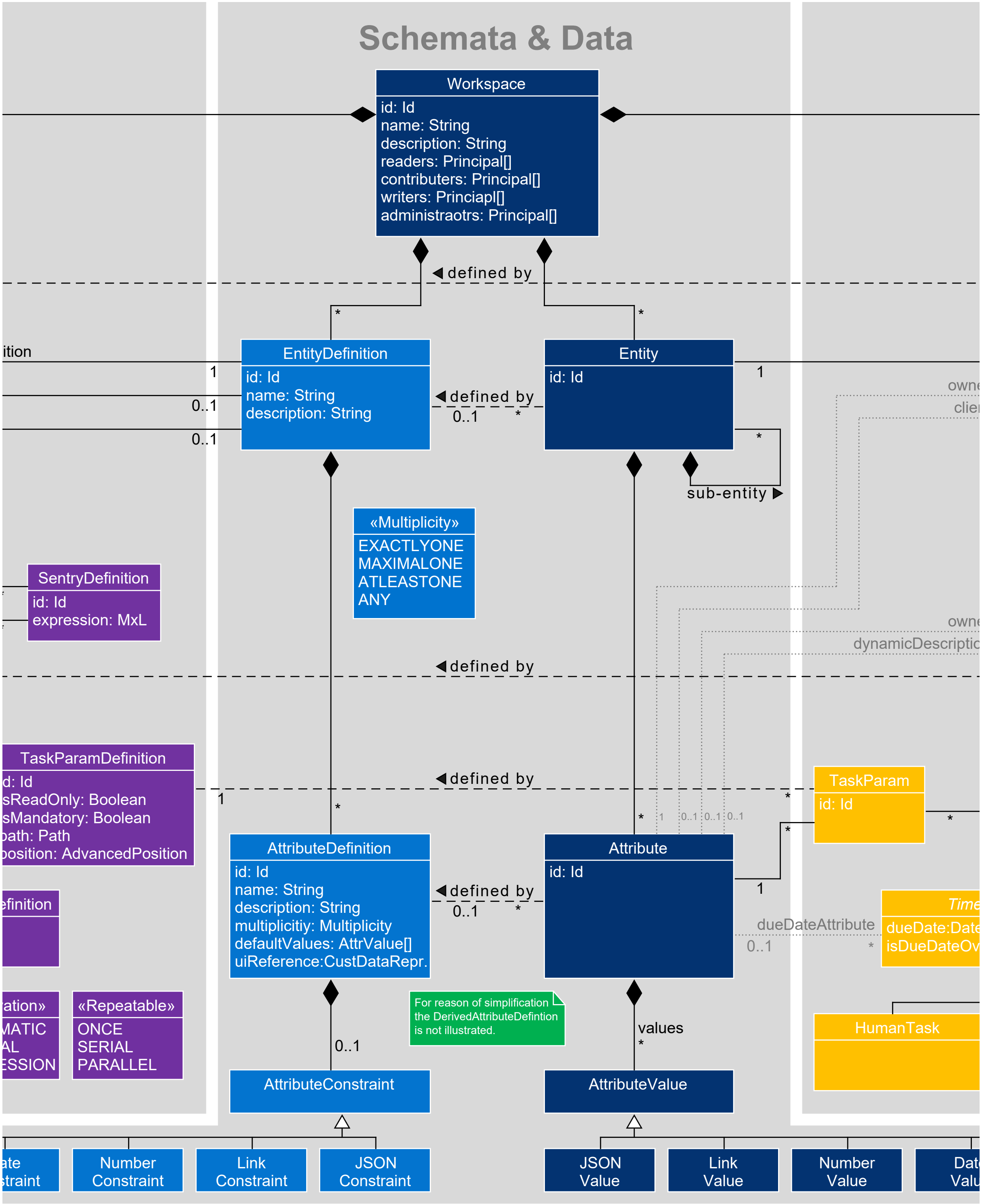


Figure A.3.: Detailed meta-model with focus on the schemata and data.

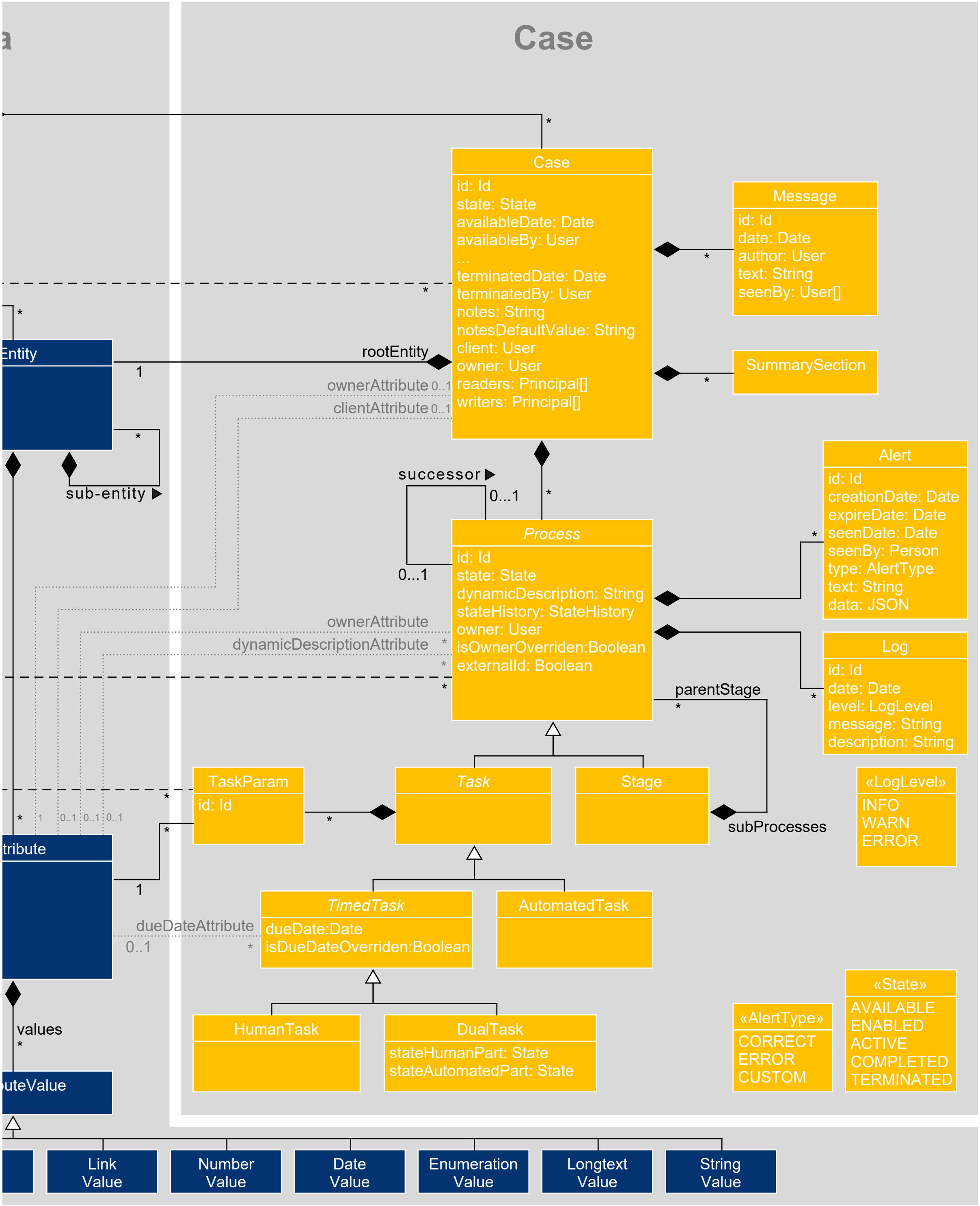


Figure A.4.: Detailed meta-model with focus on the case.

A.2. API Endpoint Reference

A.2.1. Case Execution Engine Resources

Case Resource

The Case resource contains many nested concepts that are not persisted as first-level entities and do not have a unique resource identifier. E.g., this includes the case notes, and the dynamically with each request calculated SummarySections.

GET /cases?searchQuery=Wilson&offset=0&limit=50

Returns a list of all accessible Cases across all Workspaces for the current User. The Cases contain only meta information. Additionally, query parameters are applicable to filter according to case clients and paginate the results. The presented use cases use the patient as synonymy for the client.

GET /workspaces/:id/cases?searchQuery=Wilson&offset=0&limit=50

Returns a list of all Cases within a Workspace accessible for the current User. The Cases contain only meta information. Additionally, query parameters are applicable to filter according to case clients and paginate the results.

POST /cases

Creates a Case instance based on the passed CaseDefinition id.

GET /cases/:id

Returns the Case with many details but excludes the CaseSummarySections, case team, case messages, and case notes.

GET /cases/:id/tree

Returns the Case with many details and contains all child Processes hierarchically serialized with their meta information.

DEL /cases/:id

Deletes a Case with all related resources permanently. The possible actions array encoded within the Case details indicates whether the action is allowed for the current User.

POST /cases/:id/terminate

Terminates a Case. The possible actions array encoded within the Case details indicates whether the action is allowed for the current User and case state.

POST /cases/:id/complete

Completes a Case. The possible actions array encoded within the Case details indicates whether the action is allowed for the current User and case state.

POST /cases/:id/owner/autocomplete

Returns the autocomplete options for the Case owner considering the modeled constraints.

POST /cases/:id/owner/:userId

Sets the Case owner.

`GET` `/cases/:id/team`

Returns the case team that includes the case members defining the access rights and the case roles.

`GET` `/cases/:id/team/member/autocomplete`

Returns all possible auto-complete options for a search depending on the current case members.

`POST` `/cases/:id/team/member/:principalId`

Adds or updates an existing team member. A team member is either a `User` or a `Group` and the abstraction is a `Principle`. The actual access level is passed as payload parameter.

`PATCH` `/cases/:id/team/role/:attributeId`

Sets a new `User` or `Group` (the abstraction is a `Principle`) for the role. Roles depending on the case model definition and can be added or removed. The `Attribute` resource provides the related autocomplete endpoint.

`GET` `/cases/:id/notes`

Returns the case notes rich text string and an edit token.

`PATCH` `/cases/:id/notes`

Updates the case notes if the requests edit token is equivalent with the server's edit token to prevent concurrent edits.

`GET` `/cases/:id/summarysections`

Returns the `SummarySections` for a `Case`.

`GET` `/cases/:id/processes?resourceType=humantask&state=ACTIVE`
`&possibleActions=COMPLETE&name=Charlson`

Returns a list of `Processes` related to a `Case`. Optional query parameters allow filtering according to the `resourceType`, `state`, `possibleActions`, and `name`. This endpoint is specially designed to perform the automatic execution test most efficiently.

Process Resource

The `Process` resource is an abstraction following the presented meta-model and it allows to reuse this functionality for all inherited resources.

`GET` `/process/:id/owner/autocomplete`

Returns a list of `Users` representing possible owners considering all model constraints.

Stage Resource

GET /stages/:id

Returns the Stage details.

GET /stages/:id/processes

Returns a list with all sub Processes containing only meta information.

POST /stages/:id/owner/:userId

Overrides the Stage owner. The stage's possible actions array indicates if the action can be performed in the current state by the current User.

POST /stages/:id/externalid

Sets an externalId for a Stage.

POST /stages/:id/activate

Activates a Stage. The stage's possible actions array indicates if the action can be performed in the current state by the current User.

POST /stages/:id/complete

Completes a Stage. The stage's possible actions array indicates if the action can be performed in the current state by the current User.

POST /stages/:id/terminate

Terminates a Stage. The stage's possible actions array indicates if the action can be performed in the current state by the current User.

HumanTask Resource

Several HumanTask endpoints also return DualTasks, which represent a HumanTask followed by an AutomatedTask. The common abstract concept of a HumanTask and DualTask is named TimedTask.

GET /humantasks/:id

Returns the HumanTask details.

POST /humantasks/:id/owner/:userId

Overrides the HumanTask owner. The task's possible actions array indicates if the action can be performed in the current state by the current User.

POST /humantasks/:id/duedate

Overrides the HumanTask due date. The task's possible actions array indicates if the action can be performed in the current state by the current User.

POST /humantasks/:id/externalid

Sets an externalId for a HumanTask.

POST /humantasks/:id/activate

Activates a HumanTask. The task's possible actions array indicates if the action can be performed in the current state by the current User.

POST /humantasks/:id/draft

Drafts HumanTask parameters without affecting the task's state. The task's possible actions array indicates if the action can be performed in the current state by the current User.

POST /humantasks/:id/complete

Completes a HumanTask. The task's possible actions array indicates if the action can be performed in the current state by the current User.

POST /humantasks/:id/terminate

Terminates a HumanTask. The task's possible actions array indicates if the action can be performed in the current state by the current User.

POST /humantasks/:id/correct

Corrects HumanTask parameters without affecting the task's state. The task's possible actions array indicates if the action can be performed in the current state by the current User.

GET /humantasks/me/active?isVisibleOnDashboard=true

Returns all active HumanTask across all Workspaces where the current User is assigned as owner. The result list may contain DualTasks as well because they represent a HumanTask followed by an AutomatedTask. Optionally, a query parameter allows filtering tasks according to dashboard visibility.

GET /workspace/:id/humantasks/me/active?isVisibleOnDashboard=true

Returns all active HumanTask for a certain Workspace where the current User is assigned as owner. The result list may contain DualTasks as well because they represent a HumanTask followed by an AutomatedTask. Optionally, a query parameter allows filtering tasks according to dashboard visibility.

POST /humantasks/:id/hideondashboard

Hides a TimedTask on the dashboard.

POST /humantasks/:id/unhideondashboard

Unhides a TimedTask on the dashboard.

POST /workspaces/:id/humantasks/me/active/hideondashboard

Hides all dashboard-visible TimedTasks for a certain workspace and current user on the dashboard.

POST /workspaces/:id/humantasks/me/active/unhideondashboard

Unhides all dashboard-hidden TimedTasks for a certain workspace and current user on the dashboard.

AutomatedTask Resource

GET /automatedtasks/:id

Returns the AutomatedTask details.

POST /automatedtasks/:id/owner/:userId

Overrides the AutomatedTask owner. The task's possible actions array indicates if the action can be performed in the current state by the current User.

`POST /automatedtasks/:id/externalid`

Sets an `externalId` for an `AutomatedTask`.

`POST /automatedtasks/:id/activate`

Activates an `AutomatedTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /automatedtasks/:id/draft`

Drafts `AutomatedTask` parameters without affecting the task's state. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /automatedtasks/:id/complete`

Completes an `AutomatedTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /automatedtasks/:id/terminate`

Terminates an `AutomatedTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

DualTask Resource

A `DualTasks` represents a `HumanTask` followed by an `AutomatedTask`. Therefore, the `DualTask` endpoints distinguish actions with a human part and an automated part where needed. E.g., termination while the human part is active automatically terminates the overall `DualTask` because the `AutomatedTask` depends on the `HumanTask`.

`GET /dualtasks/:id`

Returns the `DualTask` details.

`POST /dualtasks/:id/owner/:userId`

Overrides the `DualTask` owner. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/duedate`

Overrides the `DualTask` due date. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/externalid`

Sets an `externalId` for a `DualTask`.

`POST /dualtasks/:id/activate`

Activates a `DualTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/humanpart/draft`

Drafts the human part `DualTask` parameters without affecting the task's state. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/humanpart/complete`

Completes the human part of a `DualTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/humanpart/correct`

Corrects the human part `DualTask` parameters without affecting the task's state. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/automatedpart/draft`

Drafts the automated part `DualTask` parameters without affecting the task's state. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/automatedpart/complete`

Completes the automated part of a `DualTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

`POST /dualtasks/:id/terminate`

Terminates a `DualTask`. The task's possible actions array indicates if the action can be performed in the current state by the current `User`.

Message Resource

The `Messages` are marked as seen individually, meaning multiple `Users` will mark a message as seen until all case member have seen a `Message`.

`POST /messages`

Creates a `Message` the payload contains the needed `Case` relation.

`GET /messages/:id`

Returns the details for a certain `Message`.

`POST /messages/:id/seen`

Sets a `Message` as seen by the current `User` and automatically sets the seen date.

`GET /messages/me/unseen`

Returns a list with all unseen `Messages` for the current `User`.

`POST /messages/me/seen`

Sets all `Messages` as seen for the current `User` and automatically sets the seen date.

`GET /workspaces/:id/messages/me/unseen`

Returns a list with all unseen `Messages` for the current `User` for a certain `Workspace`.

`POST /workspaces/:id/messages/me/seen`

Sets all `Workspace Messages` as seen for the current `User` and automatically sets the seen date.

`GET /cases/:id/messages`

Returns a list with all case messages.

`POST /cases/:id/messages/seen`

Sets all case `Messages` as seen for the current `User` and automatically sets the seen date.

Alert Resource

For usability reasons, the user interface renamed the `Alert` concept into notifications.

`POST /alerts`

Creates an `Alert`, the related `Process` and all other parameters are contained in payload. The alert creation on the API level supports only to create `CUSTOM Alerts`. During the case execution, the engine automatically creates the `CORRECT` and `ERROR Alert` types.

`GET /alerts/:id`

Returns the details for a certain `Alert`.

`POST /alerts/:id/seen`

Sets an alert as seen and automatically sets the seen date.

`GET /alerts/me/unseen`

Returns a list with all unseen `Alerts` assigned to the current `User`.

`POST /alerts/me/seen`

Sets all `Alerts` assigned to the current `User` as seen and automatically sets the seen date.

`GET /workspaces/:id/alerts/me/unseen`

Returns a list with all unseen `Alerts` for the current `User` for a certain `Workspace`.

`POST /workspaces/:id/alerts/me/seen`

Sets all `Alerts` within a `Workspace` that are assigned to the current `User` as seen and sets the seen date automatically.

`GET /cases/:id/alerts`

Returns a list with all case-related `Alerts`.

`POST /cases/:id/alerts/seen`

Sets all case `Alerts` as seen and automatically sets the seen date.

`GET /processes/:id/alerts`

Returns a list with all process-related `Alerts`.

Log Resource

`Logs` are generated by the execution engine and are immutable on the API level. A typical sample for a `Log` creation is the execution of hooks. It contains the HTTP status code, the possibly occurring stack trace, and the serialized `Process` payload data.

GET /cases/:id/logs

Returns a list of all Case related Logs.

GET /processes/:id/logs

Returns a list of all Process related Logs. A Process resource is an abstraction for the Stage, HumanTask, DualTask, AutomatedTask resource.

A.2.2. Case-Based Process Model Resources

CaseDefinition Resource

POST /casedefinitions

Creates a CaseDefinition.

GET /casedefinitions/:id

Returns a CaseDefinition.

PATCH /casedefinitions/:id

Updates a CaseDefinition.

DEL /casedefinitions/:id

Deletes a CaseDefinition including all associated SummarySectionDefinitions and ProcessDefinitions and instantiated Cases.

GET /casedefinitions/:id/tree

Returns all CaseDefinitions including all ProcessDefinitions hierarchically serialized.

GET /casedefinitions/caninstantiate

Returns all Cases across Workspaces that can be instantiated by the current User.

GET /workspaces/:id/casedefinitions/caninstantiate

Returns all Cases for the Workspaces that can be instantiated by the current User.

GET /workspaces/:id/casedefinitions

Returns all CaseDefinitions contained within the Workspace.

GET /casedefinitions/version

Returns a list containing the CaseDefinition name and the associated version.

StageDefinition Resource

POST /humantaskdefinitions

Creates a StageDefinition that is either a direct child of the CaseDefinition or a sub process definition of a StageDefinition.

GET /humantaskdefinitions/:id

Returns a StageDefinition.

`PATCH   /humantaskdefinitions/:id`

Updates a `StageDefinition`.

`DEL     /humantaskdefinitions/:id`

Deletes a `StageDefinition` including all associated `HttpHookDefinitions` and `SentryDefinitions` and sub process definitions including `StageDefinitions`, `HumanTaskDefinitons`, `DualTaskDefinitions` and `AutomatedTaskDefinitions`.

HumanTaskDefinition Resource

`POST    /humantaskdefinitions`

Creates a `HumanTaskDefinition` that is either a direct child of the `CaseDefinition` or a sub process definition of a `StageDefinition`.

`GET     /humantaskdefinitions/:id`

Returns a `HumanTaskDefinition`.

`PATCH   /humantaskdefinitions/:id`

Updates a `HumanTaskDefinition`.

`DEL     /humantaskdefinitions/:id`

Deletes a `HumanTaskDefinition`, including all associated `HttpHookDefinitions` and `SentryDefinitions`.

AutomatedTaskDefinition Resource

`POST    /automatedtaskdefinitions`

Creates an `AutomatedTaskDefinition` that is either a direct child of the `CaseDefinition` or a sub process definition of a `StageDefinition`.

`GET     /automatedtaskdefinitions/:id`

Returns an `AutomatedTaskDefinition`.

`PATCH   /automatedtaskdefinitions/:id`

Updates an `AutomatedTaskDefinition`.

`DEL     /automatedtaskdefinitions/:id`

Deletes an `AutomatedTaskDefinition` including all associated `HttpHookDefinitions` and `SentryDefinitions`.

DualTaskDefinition Resource

POST /dualtaskdefinitions

Creates a DualTaskDefinition that is either a direct child of the CaseDefinition or a sub process definition of a StageDefinition.

GET /dualtaskdefinitions/:id

Returns a DualTaskDefinition.

PATCH /dualtaskdefinitions/:id

Updates a DualTaskDefinition.

DEL /dualtaskdefinitions/:id

Deletes a DualTaskDefinition, including all associated HttpHookDefinitions and SentryDefinitions.

TaskParamDefinition Resource

POST /taskparamdefinitions

Creates a TaskParameterDefinition for a ProcessDefinition that is encoded in the payload.

GET /taskparamdefinitions/:id

Returns a TaskParameterDefinition.

PATCH /taskparamdefinitions/:id

Updates a TaskParameterDefinition.

DEL /taskparamdefinitions/:id

Deletes a TaskParamDefinition.

GET /taskdefinitions/:id/taskparamdefinitions*

Returns all TaskParamDefinitions for the related ProcessDefinition.

SentryDefinition Resource

POST /sentrydefinitions

Creates a SentryDefinition for a ProcessDefinition that is encoded in the payload.

GET /sentrydefinitions/:id

Returns a SentryDefinition.

PATCH /sentrydefinitions/:id

Updates a SentryDefinition.

DEL /sentrydefinitions/:id

Deletes a SentryDefinition.

`GET` `/processdefinitions/:id/sentrydefinitions`

Returns all `SentryDefinitions` for the related `ProcessDefinition`.

SummarySectionDefinition Resource

`POST` `/summarysectiondefinitions`

Creates a `SummarySectionDefinition` for a `CaseDefinition` that is encoded in the payload.

`GET` `/summarysectiondefinitions/:id`

Returns a `SummarySectionDefinition`.

`PATCH` `/summarysectiondefinitions/:id`

Updates a `SummarySectionDefinition`.

`DEL` `/summarysectiondefinitions/:id`

Deletes a `SummarySectionDefinition`.

`GET` `/casedefinitions/:id/summarysectiondefinitions*`

Returns all `SummarySectionDefinitions` for the related `CaseDefinition`.

HttpHookDefinition Resource

`POST` `/httphookdefinitions`

Creates a `HttpHookDefinition` for a `ProcessDefinition` that is encoded in the payload.

`GET` `/httphookdefinitions/:id`

Returns a `HttpHookDefinition`.

`PATCH` `/httphookdefinitions/:id`

Updates a `HttpHookDefinition`.

`DEL` `/httphookdefinitions/:id`

Deletes a `HttpHookDefinition`.

`GET` `/processdefinitions/:id/httphookdefinitions`

Returns all `HttpHookDefinitions` for the related `ProcessDefinition`.

A.2.3. Higher-Order Functional Language Resources

🟩 DerivedAttributeDefinition Resource

`POST /derivedattributedefinitions`*

Creates a `DerivedAttributeDefinition`, the payload contains an association to the related `Entity Definition`.

`GET /derivedattributedefinitions/:id`*

Returns a `DerivedAttributeDefinition`.

`PATCH /derivedattributedefinitions/:id`*

Updates a `DerivedAttributeDefinition`.

`DEL /derivedattributedefinitions/:id`*

Deletes a `DerivedAttributeDefinition`.

A.2.4. Role-Based and Discretionary Access Control Model Resources

🟥 UserDefinition Resource

`GET /userdefinitions`

Returns the `UserDefinition` which is basically an `EntityDefinition` with a specific id. This endpoint is only exposed to get the id of the user's `EntityDefinition`, all update operation are performed based on the `EntityDefinition` resource.

🟥 User Resource

`POST /users`

Creates a new `User`.

`GET /users`

Returns all `Users`.

`GET /users/:id`

Returns the `User` with the `id`.

`GET /users/me`

Returns the current `User` object.

`POST /users/:id/disable`

Disables the `User` with the `id`.

`POST /users/:id/enable`

Enables the `User` with the `id`.

`PATCH /users/:id`

Updates the User with the id.

`DEL /users/:id`

Deletes the User with the id.

Group Resource

`POST /groups`

Creates a new Group.

`GET /groups`

Returns all Groups.

`GET /groups/:id`

Returns the Group with the id.

`PATCH /groups/:id`

Updates the Group with the id.

`DEL /groups/:id`

Deletes the Group with the id.

`POST /groups/:id/member/:principalId`

Adds a Group member with the princiaplId. A Principal is either a User or Group.

`DEL /groups/:id/member/:principalId`

Removes a Group member with the princiaplId. A Principal is either a User or Group.

A.2.5. Multiple Dynamic Schemata Resources

EntityDefinition Resource

`POST /entitydefinitions*`

Creates an EntityDefinition.

`GET /entitydefinitions/:id*`

Returns an EntityDefinition.

`PATCH /entitydefinitions/:id*`

Updates an EntityDefinition.

`DEL /entitydefinitions/:id*`

Deletes an EntityDefinition.

Attribute Definition Resource

POST /attributedefinitions*

Creates an AttributeDefinition, the payload contains an association to the related EntityDefinition.

GET /attributedefinitions/:id*

Returns an AttributeDefinition.

PATCH /attributedefinitions/:id*

Updates an AttributeDefinition.

DEL /attributedefinitions/:id*

Deletes an AttributeDefinition.

A.2.6. Annotated Versioned Linked Graph Resources

Workspace Resource

GET /workspaces

Returns all Workspaces where the current user has read access rights.

POST /workspaces

Creates a Workspaces.

GET /workspaces/:id

Returns a Workspace.

PATCH /workspaces/:id*

Updates a Workspace.

DEL /workspaces/:id

Deletes a Workspace with all its containing elements.

Entity Resource

GET /entities/:id

Returns a detailed Entity representation containing the related attributes.

GET /entities/:id/navigationtree

Returns a navigation tree based on AttributeValues linking to Entities. Occurring recursive linkages are ignored, only the first occurrence is considered.

Attribute Resource

GET /attributes/:id/autocomplete

Returns all possible values for a certain Attribute considering all modeled constraints.

A.2.7. Administrative Resources

Import Resource

The file must be attached as a binary body and it must not contain the `Content-Type: application/json`. Please note this request may take several minutes, depending on the specified models.

`POST /import/workspaces`

Imports `Workspaces` initial `Groups` and `Users` from an XML file.

`POST /import/casedefinitions?version=11&isExecute=true&isDebug=false`

Imports an XML case template file. The `version` parameter allows associating the uploaded model with a specific version number. If a model contains a test declaration that can be automatically executed, the parameter `isExecute` should be set to `true` to validate the model with the declared test workflow. The execution uses the current user. If that is not sufficient due to model constraints an `Execution-User` can be defined similar to the `Simulate-User` header parameter. The `isDebug` parameter pauses the test execution at each declared breakpoint within the XML file and the execution is continued by pressing enter on the console of the server. The execution and debug option should only be used in a test environment.

Settings Resource

`GET /settings*`

Returns the current settings that includes which `Principal` is allowed to create or edit a `Workspace`, a `Group`, a `User` and `UserDefinition`.

`PATCH /settings*`

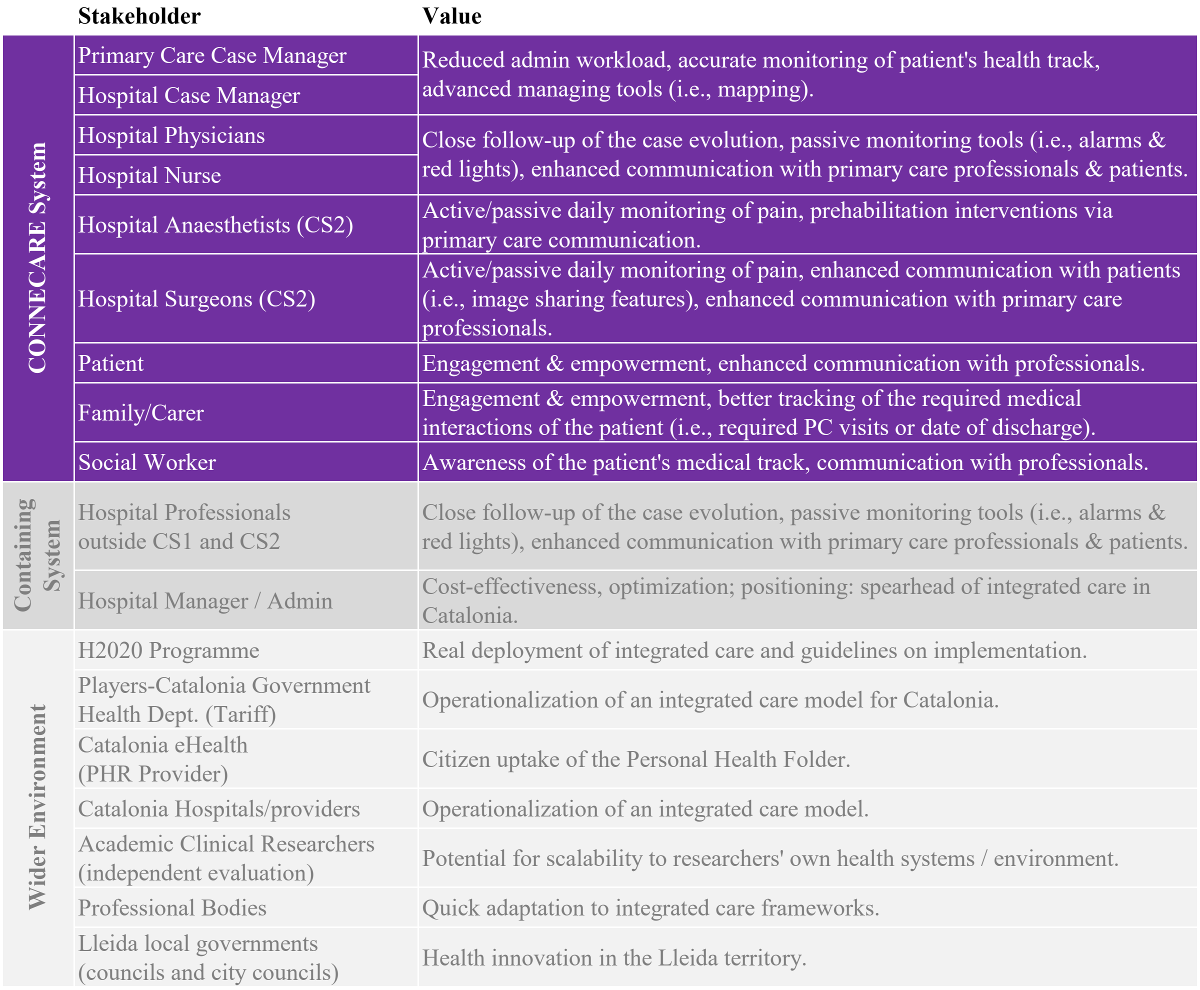
Updates the settings.

Context Resource

`GET /`

Provides contextual information about the current build version that is useful for debugging purposes. The build date, related commit hash and the docker tag are provided. Additionally, a timestamp indicates when the server was started.

A.3. Implementation Study Case Studies



	Stakeholder	Value
CONNECARE System	Primary Care Case Manager	Reduced admin workload, accurate monitoring of patient's health track, advanced managing tools (i.e., mapping).
	Hospital Case Manager	
	Hospital Physicians	Close follow-up of the case evolution, passive monitoring tools (i.e., alarms & red lights), enhanced communication with primary care professionals & patients.
	Hospital Nurse	
	Hospital Anaesthetists (CS2)	Active/passive daily monitoring of pain, prehabilitation interventions via primary care communication.
	Hospital Surgeons (CS2)	Active/passive daily monitoring of pain, enhanced communication with patients (i.e., image sharing features), enhanced communication with primary care professionals.
	Patient	Engagement & empowerment, enhanced communication with professionals.
	Family/Carer	Engagement & empowerment, better tracking of the required medical interactions of the patient (i.e., required PC visits or date of discharge).
	Social Worker	Awareness of the patient's medical track, communication with professionals.
Containing System	Hospital Professionals outside CS1 and CS2	Close follow-up of the case evolution, passive monitoring tools (i.e., alarms & red lights), enhanced communication with primary care professionals & patients.
	Hospital Manager / Admin	Cost-effectiveness, optimization; positioning: spearhead of integrated care in Catalonia.
Wider Environment	H2020 Programme	Real deployment of integrated care and guidelines on implementation.
	Players-Catalonia Government Health Dept. (Tariff)	Operationalization of an integrated care model for Catalonia.
	Catalonia eHealth (PHR Provider)	Citizen uptake of the Personal Health Folder.
	Catalonia Hospitals/providers	Operationalization of an integrated care model.
	Academic Clinical Researchers (independent evaluation)	Potential for scalability to researchers' own health systems / environment.
	Professional Bodies	Quick adaptation to integrated care frameworks.
	Lleida local governments (councils and city councils)	Health innovation in the Lleida territory.

Table A.1.: Stakeholders value in Lleida adapted from CONNECARE Consortium (2019b).

A.4. Implementation Study Case Templates

	Groningen		Tel Aviv		Lleida	
	CS1	CS2	CS1	CS2	CS1	CS2
Monitoring Prescription	◐	●	◐	◐	◐	◐
├ Blood Pressure	●	●	●	●	●	●
├ Body Temperature	●	●	○	○	○	○
├ Weight	○	●	○	○	●	○
└ Heart Rate	○	●	○	○	●	○
└ Oxygen Saturation with O2	○	○	○	○	●	○
└ Oxygen Saturation without O2	○	○	○	○	●	○
Drug Prescription	◐	◐	◐	◐	◐	◐
Physical Activity	●	●	●	●	●	●
Patient Questionnaire	◐	◐	◐	◐	◐	◐
├ SF-12	●	●	●	●	○	○
├ EQ5D	●	●	●	●	○	○
├ AHS post-chirurgical	○	●	○	○	●	●
├ S-LANSS	○	○	○	○	○	●
├ AHS EPOC	○	○	○	○	●	○
├ AHS IC	○	○	○	○	●	○
├ Verbal Numerical Rating Scale	○	○	○	○	○	●
├ WOMAC	○	○	○	○	○	●
├ NPS	○	●	○	○	○	○
├ SUS	○	●	○	○	○	○
├ VAS	○	●	○	○	○	○
├ LEAVE	○	○	●	●	○	○
├ EAT	○	○	●	●	○	○
├ FEEL	○	○	●	●	○	○
├ DRINK	○	○	●	●	○	○
├ SATPACASSUTA	○	○	●	●	○	○
├ SUSASSUTA	○	○	●	●	○	○
├ P3CEQ	○	○	●	●	○	○
├ NCQ	○	○	●	●	○	○
├ CCQ	●	○	○	○	○	○
├ ACQ	●	○	○	○	○	○
└ CARAT	●	○	○	○	○	○
Simple Task	○	○	●	●	○	○
├ Read book or newspaper	○	○	●	●	○	○
├ Walk slowly outside	○	○	●	●	○	○
├ ... 23 more specific ...	○	○	●	●	○	○
└ Other	○	○	●	●	○	○
Advice	●	●	●	●	●	●

○ not applied ◐ partly applied ● fully applied

Table A.2.: Extended DualTask system integration and orchestration models representing each a corresponding micro-service within the SMS.

A.4.1. Case Templates Groningen

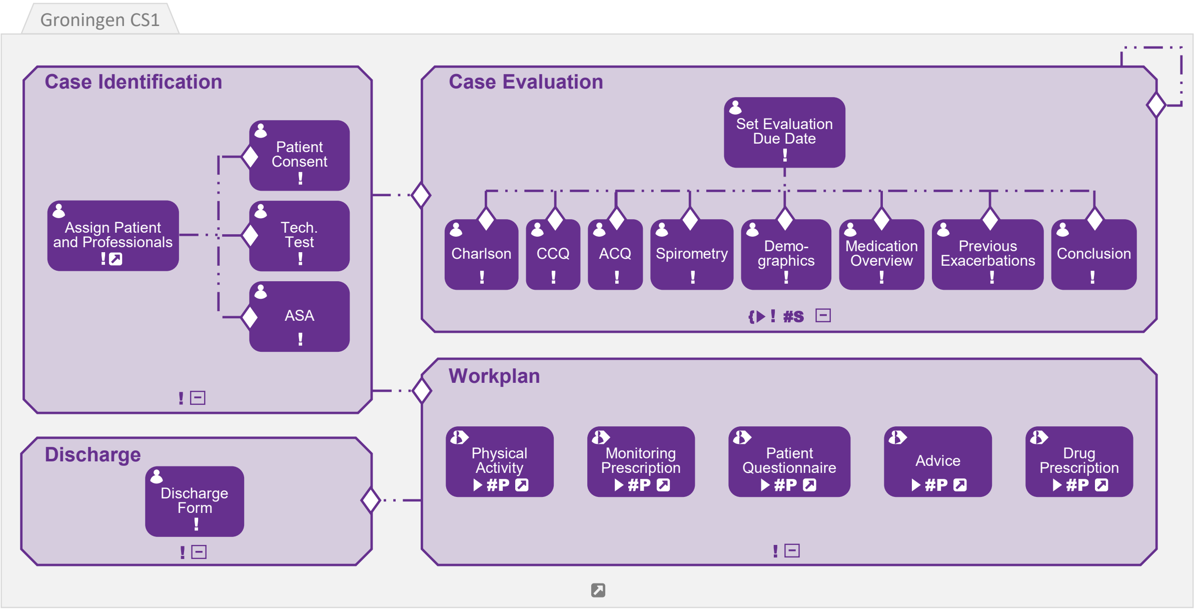


Figure A.5.: Case template Groningen CS1.

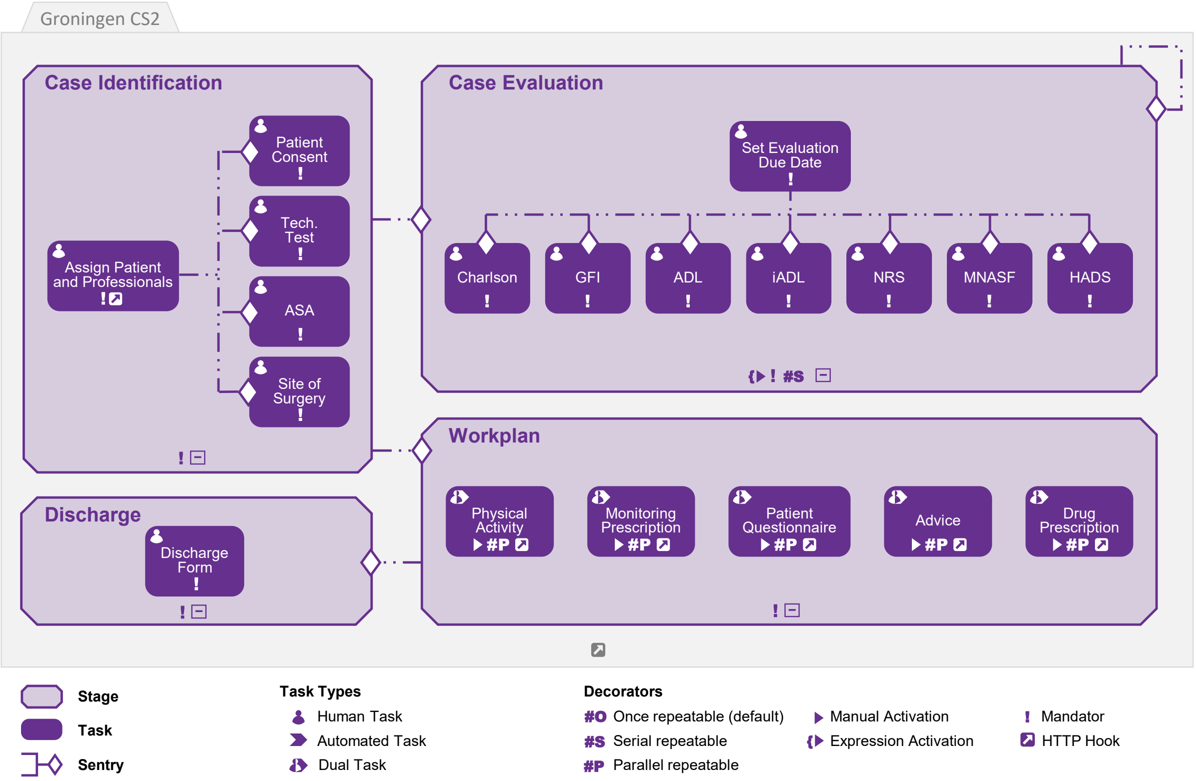


Figure A.6.: Case template Groningen CS2.

A.4.2. Case Templates Tel Aviv

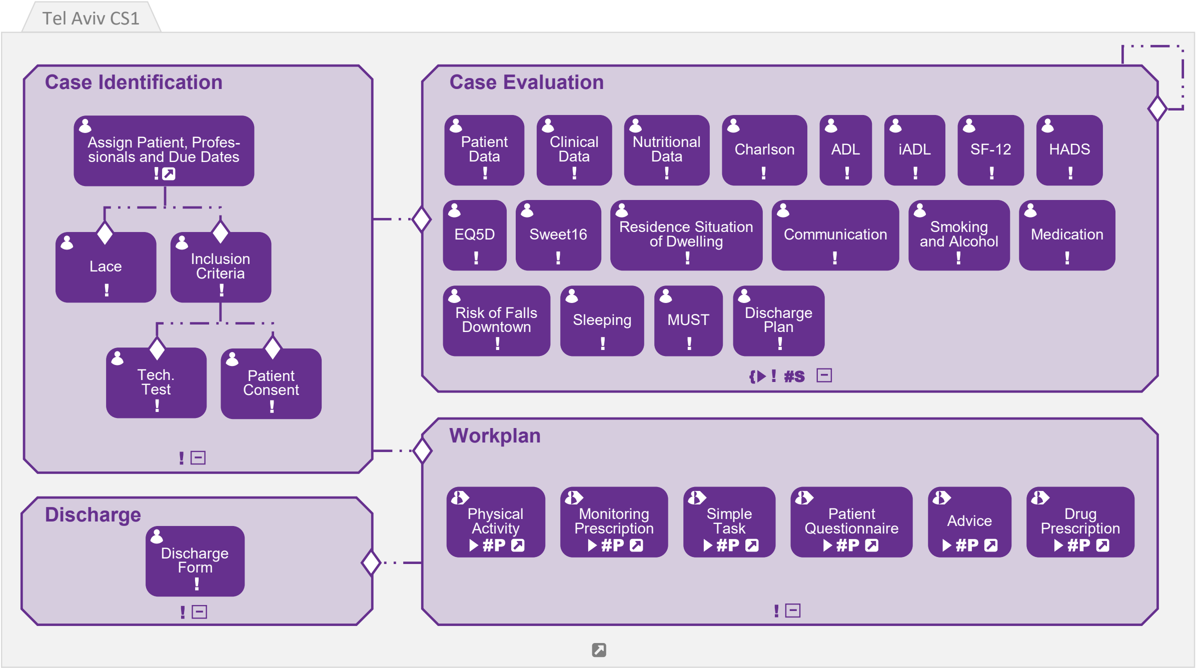


Figure A.7.: Case template Tel Aviv CS1.

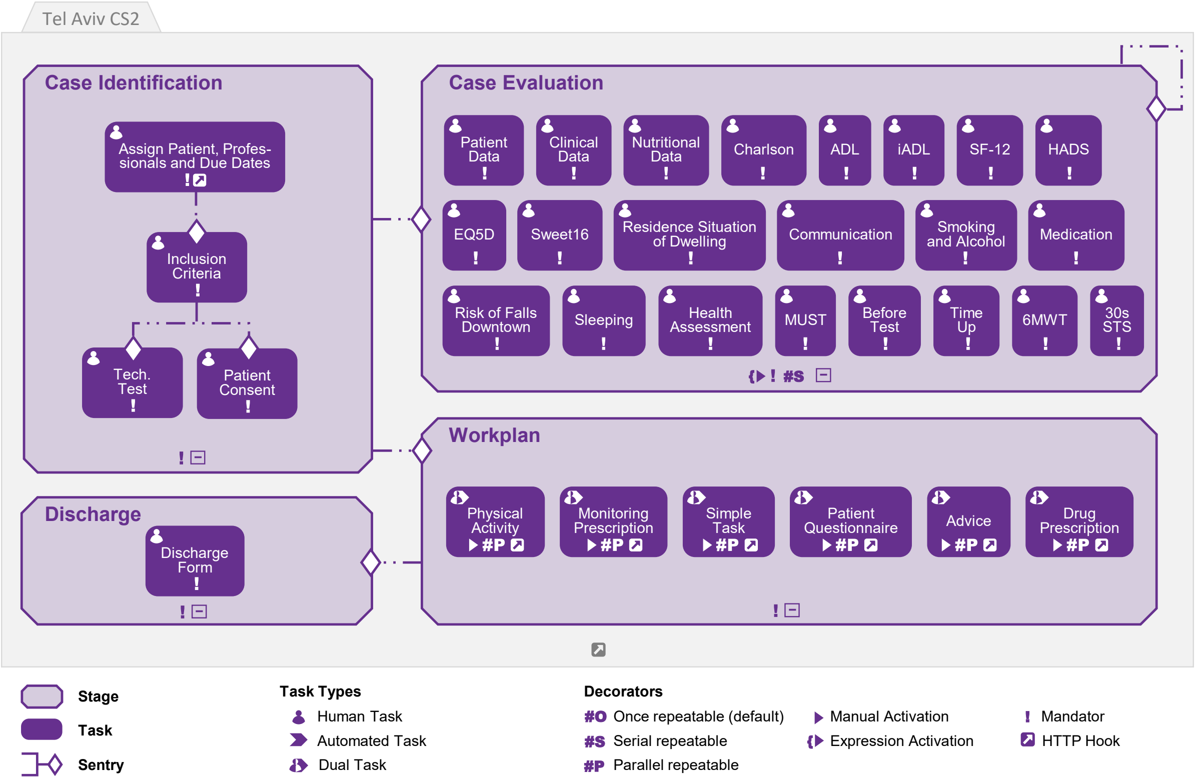


Figure A.8.: Case template Tel Aviv CS2.

A.4.3. Case Templates Lleida

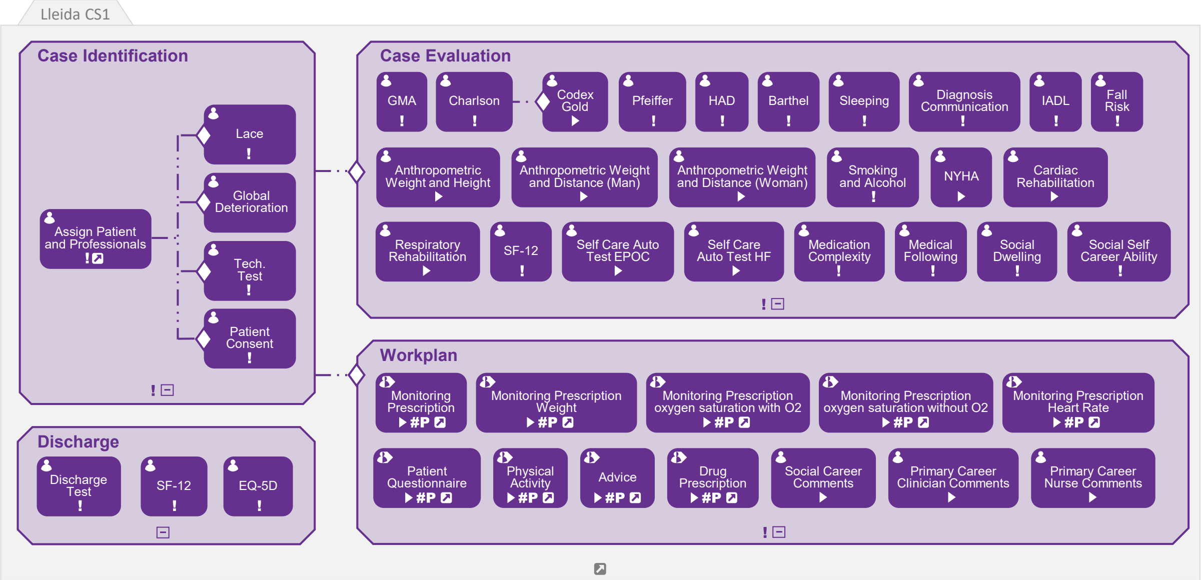


Figure A.9.: Case template Lleida CS1.

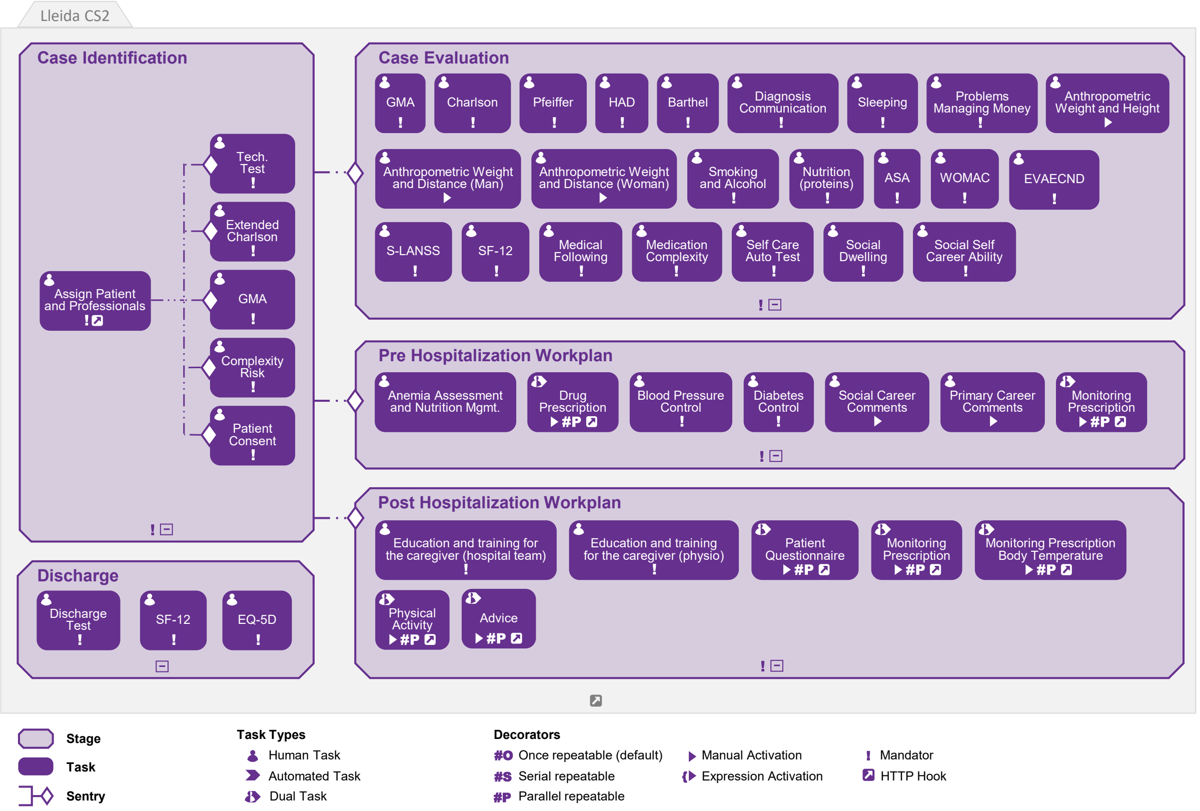
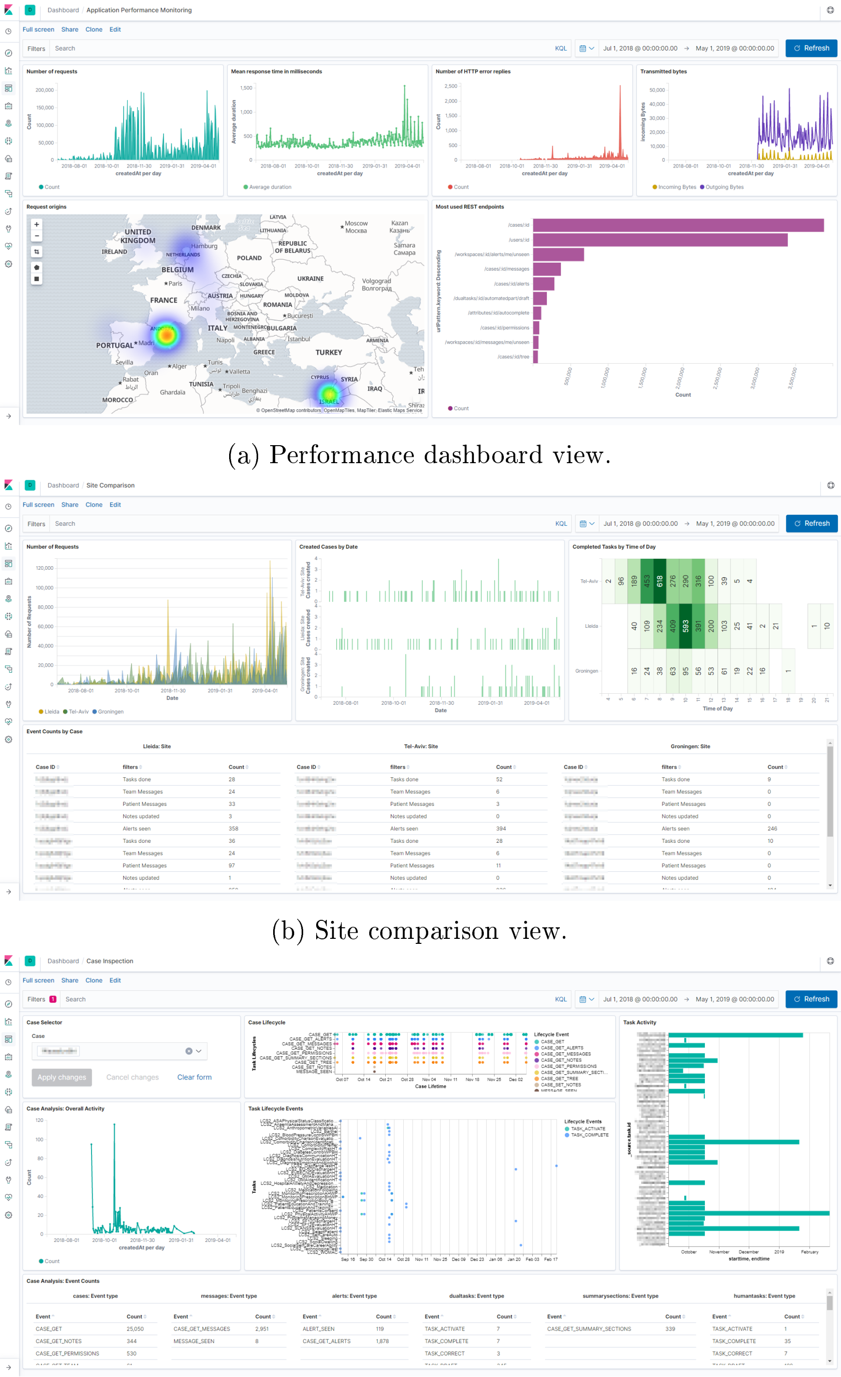


Figure A.10.: Case template Lleida CS2.

A.5. Implementation Study Case Execution Behavior



Dashboard / Application Performance Monitoring
Full screen Share Clone Edit
Filters Search KQL Jul 1, 2018 @ 00:00:00.00 → May 1, 2019 @ 00:00:00.00 Refresh
Number of requests
Mean response time in milliseconds
Number of HTTP error replies
Transmitted bytes
createdAt per day
Count
Average duration
Incoming Bytes
Outgoing Bytes
Request origins
Most used REST endpoints
/cases/:id
/users/:id
/workspaces/:id/alerts/me/unseen
/cases/:id/messages
/cases/:id/alerts
/dualtasks/:id/automatedpart/draft
/attributes/:id/autocomplete
/cases/:id/permissions
/workspaces/:id/messages/me/unseen
/cases/:id/tree
urlPattern.keyword: Descending
© OpenStreetMap contributors, OpenMapTiles, MapTiler, Elastic Maps Service

(a) Performance dashboard view.

Dashboard / Site Comparison
Full screen Share Clone Edit
Filters Search KQL Jul 1, 2018 @ 00:00:00.00 → May 1, 2019 @ 00:00:00.00 Refresh
Number of Requests
Created Cases by Date
Completed Tasks by Time of Day
Date
Time of Day
Lleida
Tel-Aviv
Groningen

Event Counts by Case

Lleida: Site

Case ID	filters	Count
[redacted]	Tasks done	28
[redacted]	Team Messages	24
[redacted]	Patient Messages	33
[redacted]	Notes updated	3
[redacted]	Alerts seen	358
[redacted]	Tasks done	36
[redacted]	Team Messages	24
[redacted]	Patient Messages	97
[redacted]	Notes updated	1

Tel-Aviv: Site

Case ID	filters	Count
[redacted]	Tasks done	52
[redacted]	Team Messages	6
[redacted]	Patient Messages	3
[redacted]	Notes updated	0
[redacted]	Alerts seen	394
[redacted]	Tasks done	28
[redacted]	Team Messages	6
[redacted]	Patient Messages	11
[redacted]	Notes updated	0

Groningen: Site

Case ID	filters	Count
[redacted]	Tasks done	9
[redacted]	Team Messages	0
[redacted]	Patient Messages	0
[redacted]	Notes updated	0
[redacted]	Alerts seen	246
[redacted]	Tasks done	10
[redacted]	Team Messages	0
[redacted]	Patient Messages	0
[redacted]	Notes updated	0

(b) Site comparison view.

Case Analysis: Event Counts

cases: Event type

Event	Count
CASE_GET	25,050
CASE_GET_NOTES	344
CASE_GET_PERMISSIONS	530

messages: Event type

Event	Count
CASE_GET_MESSAGES	2,951
MESSAGE_SEEN	8

alerts: Event type

Event	Count
ALERT_SEEN	119
CASE_GET_ALERTS	1,878

dualtasks: Event type

Event	Count
TASK_ACTIVATE	7
TASK_COMPLETE	7
TASK_CORRECT	3

summarysections: Event type

Event	Count
CASE_GET_SUMMARY_SECTIONS	339

humantasks: Event type

Event	Count
TASK_ACTIVATE	1
TASK_COMPLETE	35
TASK_CORRECT	7

(c) Case inspection view.

Figure A.11.: Kibana screens used for log and pattern inspection (Bönisch, 2019).

A.5.1. Case Execution Behavior in Groningen

	CS1					CS2				
Indicator	**Identification**	**Evaluation**	**Workplan**	**Discharge**	**Case Level**	**Identification**	**Evaluation**	**Workplan**	**Discharge**	**Case Level**
Number of cases	25	1	16	0	25	40	4	42	0	43
Number of activities	4	1	2	0	6	5	8	5	0	8
Number of overall paths	3	0	3	0	14	6	13	20	0	20
Mean number of paths per activity	1.5	0	2	-	4	2.4	3.3	7	-	4.4
Number of process variants	1	1	5	-	19	3	3	41	-	42
Maximum share of cases per variant	100%	100	69%	-	16%	88%	50%	5%	-	5%
Mean share of manual activated tasks	0%	0%	100%	0%	7%	0%	0%	100%	0%	24%

Table A.3.: Case execution characteristics and run-time planning in Groningen.

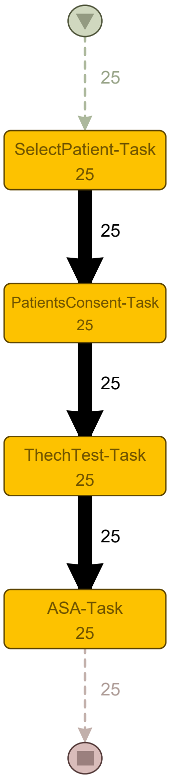


Figure A.12.: Case execution Groningen CS1 Identification stage.

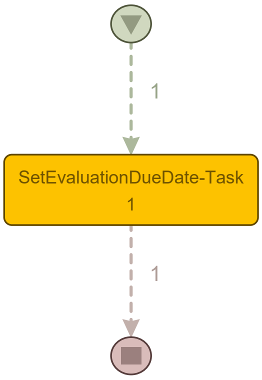


Figure A.13.: Case execution Groningen CS1 Evaluation stage.

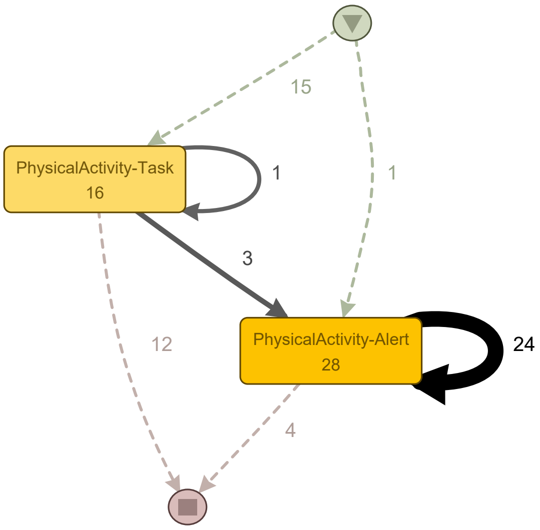


Figure A.14.: Case execution Groningen CS1 Workplan stage.

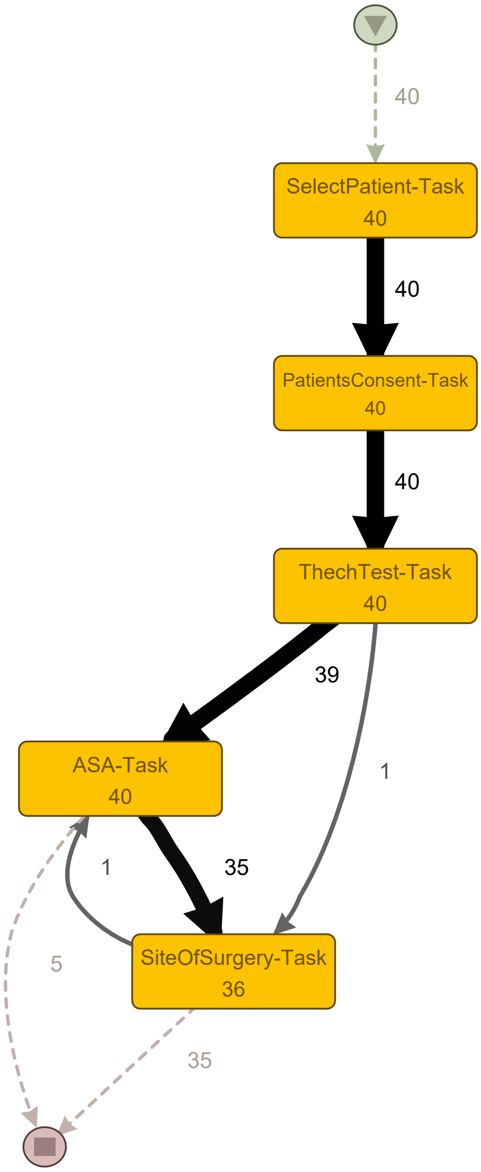


Figure A.15.: Case execution Groningen CS2 Identification stage.

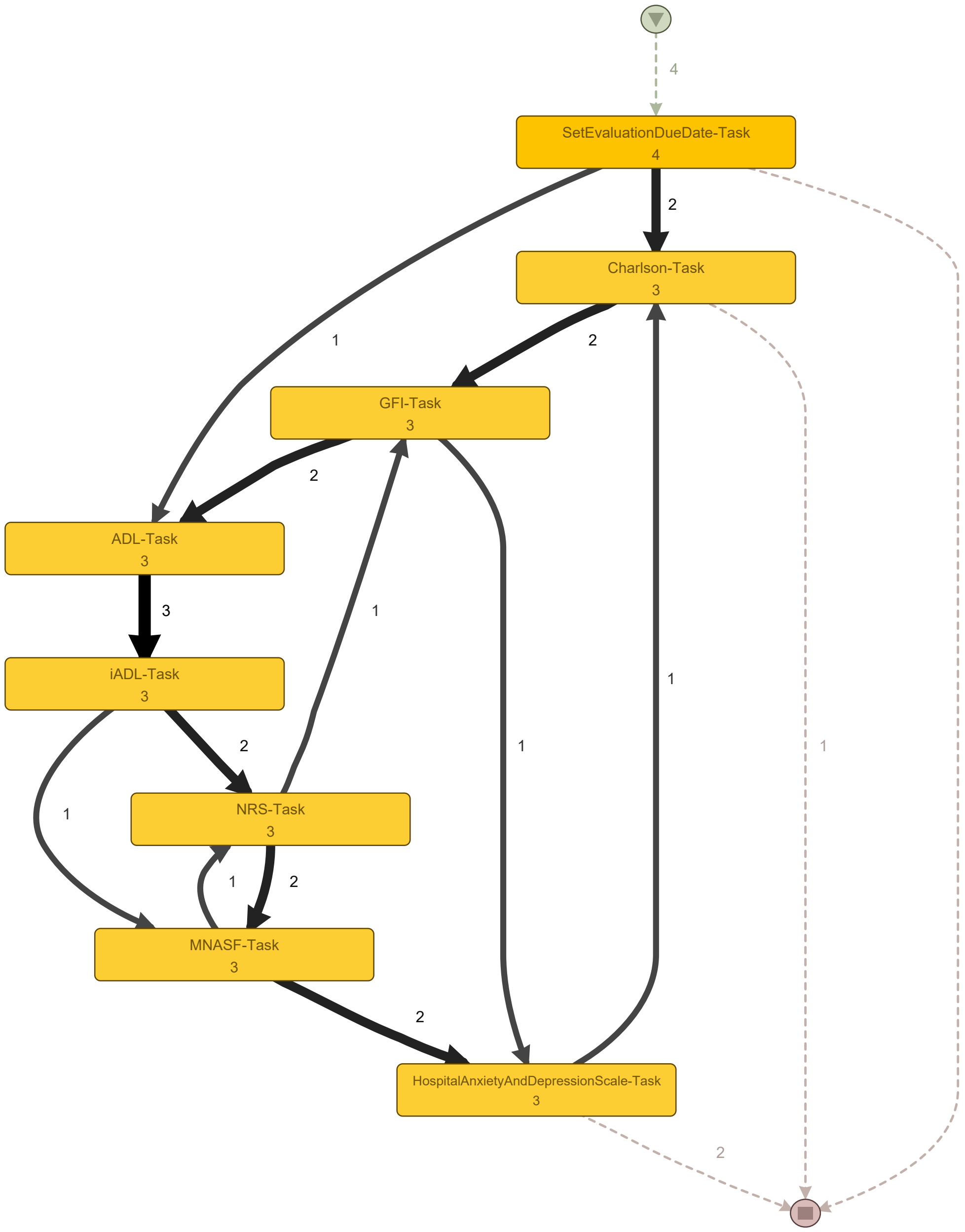


Figure A.16.: Case execution Groningen CS2 Evaluation stage.

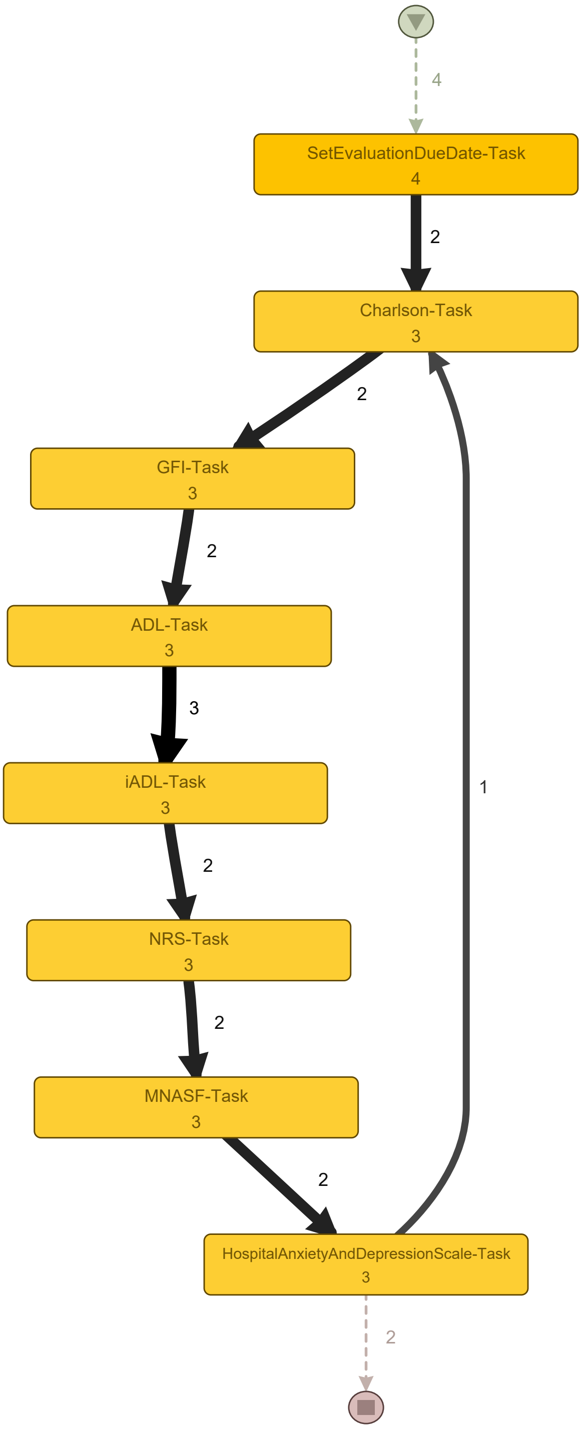


Figure A.17.: Case execution Groningen CS2 Evaluation stage with abstracted paths.

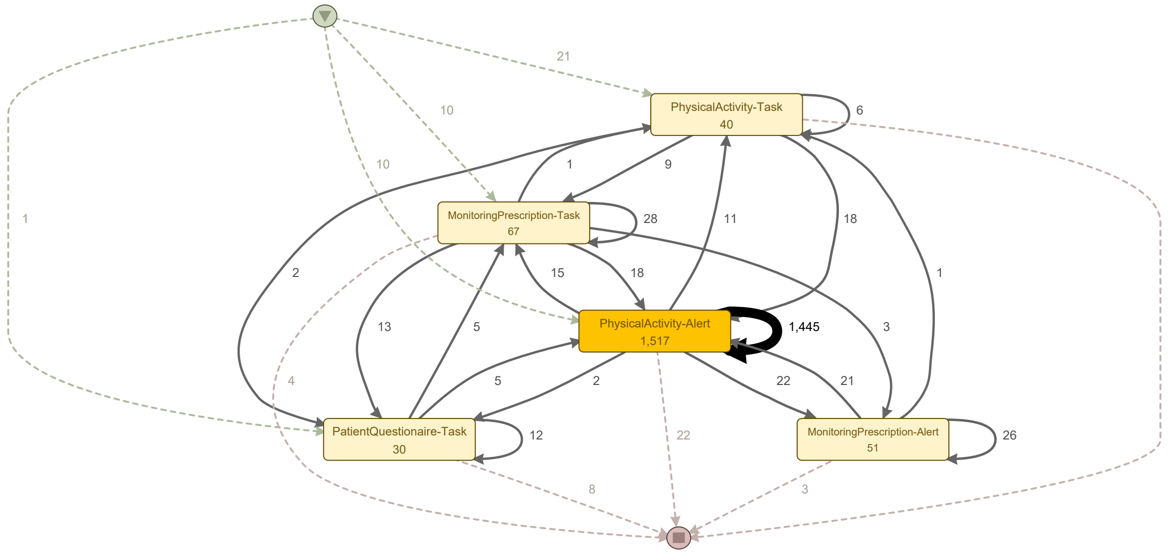


Figure A.18.: Case execution Groningen CS2 Workplan stage.

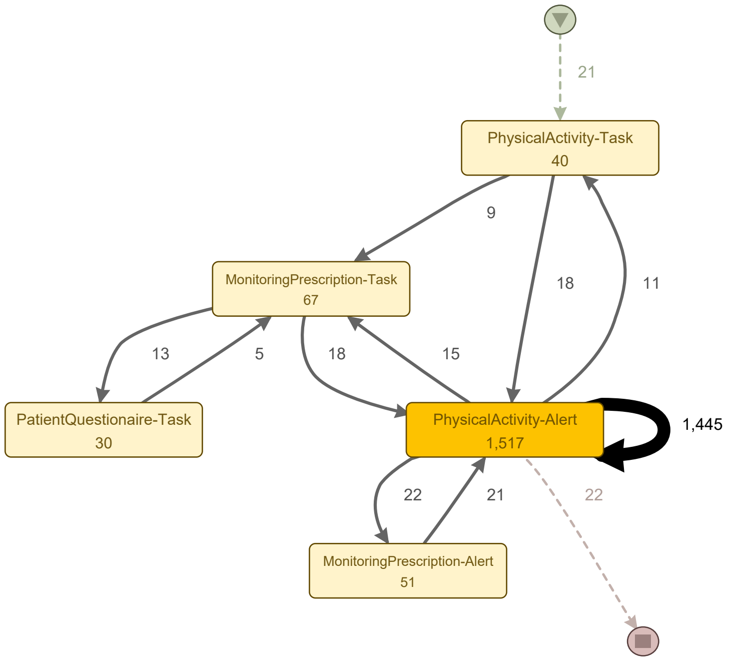


Figure A.19.: Case execution Groningen CS2 Workplan stage with abstracted paths.

A.5.2. Case Execution Behavior in Tel Aviv

	CS1					CS2				
Indicator	**Identification**	**Evaluation**	**Workplan**	**Discharge**	**Case Level**	**Identification**	**Evaluation**	**Workplan**	**Discharge**	**Case Level**
Number of cases	54	50	56	1	57	30	32	32	0	32
Number of activities	5	18	7	1	9	4	22	7	0	9
Number of overall paths	9	101	32	0	44	6	115	27	0	38
Mean number of paths per activity	3.6	11.2	8.1	0	9.1	3	10.4	6.7	-	7.8
Number of process variants	5	36	47	1	56	2	29	32	-	32
Maximum share of cases per variant	81%	20%	9%	100%	4%	97%	6%	3%	-	3%
Mean share of manual activated tasks	0%	0%	100%	0%	12%	0%	0%	100%	0%	17%

Table A.4.: Case execution characteristics and run-time planning Tel Aviv.

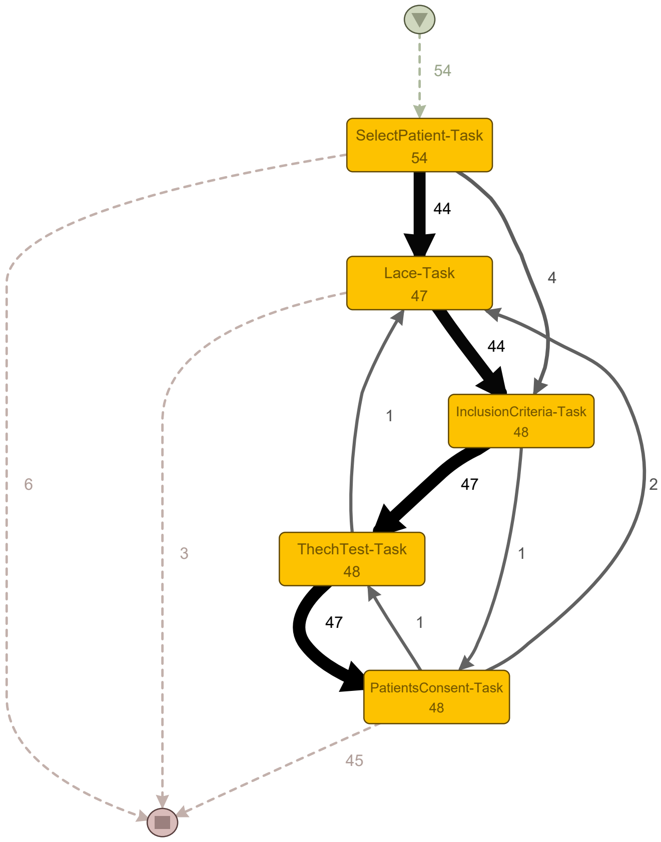


Figure A.20.: Case execution Tel Aviv CS1 Identification stage.

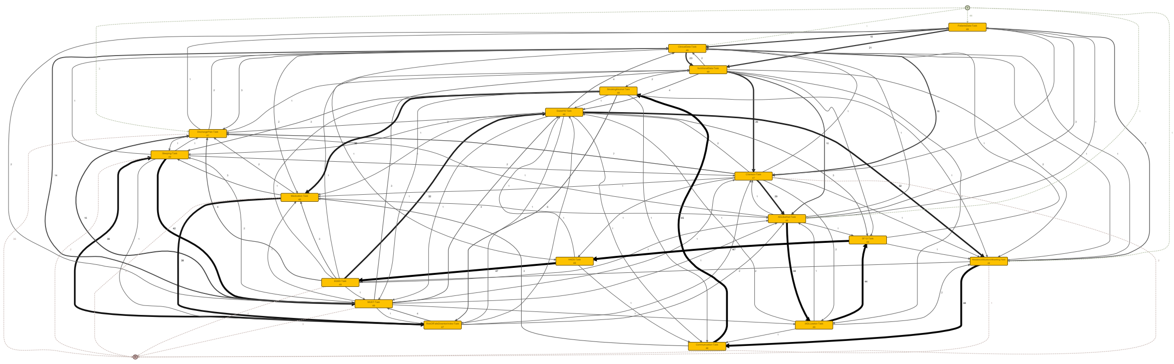


Figure A.21.: Case execution Tel Aviv CS1 Evaluation stage.

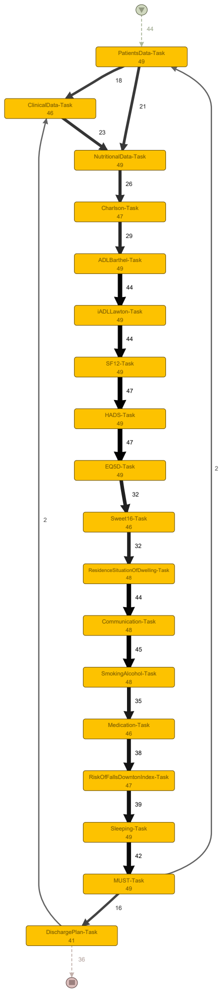


Figure A.22.: Case execution Tel Aviv CS1 Evaluation stage with abstracted paths.

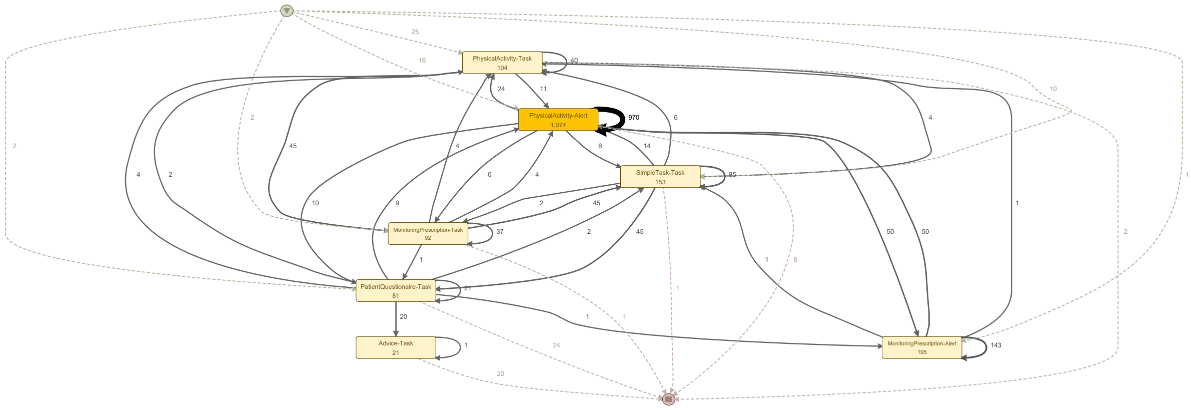


Figure A.23.: Case execution Tel Aviv CS1 Workplan stage.

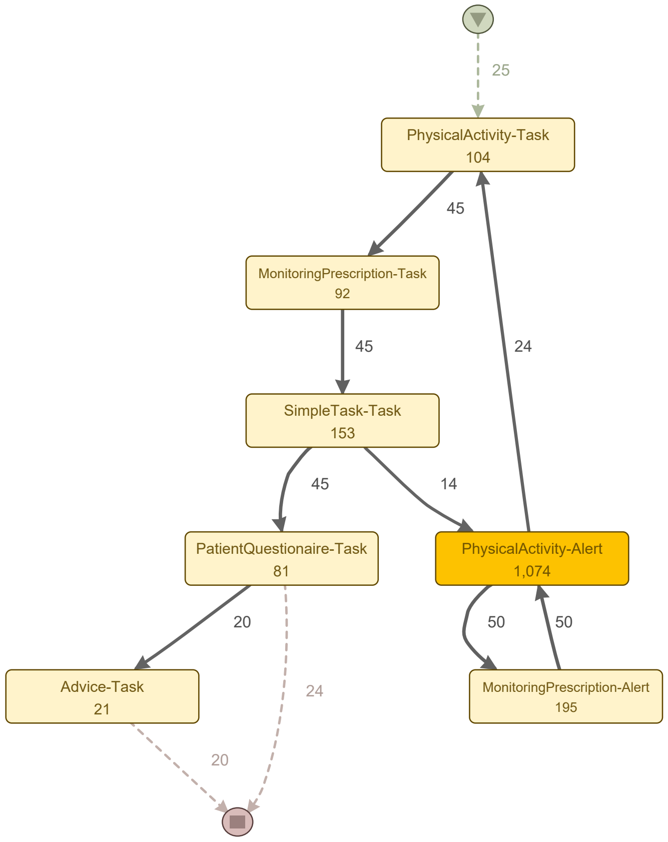


Figure A.24.: Case execution Tel Aviv CS1 Workplan stage with abstracted paths.

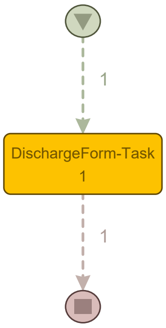


Figure A.25.: Case execution Tel Aviv CS1 Discharge stage with abstracted paths.

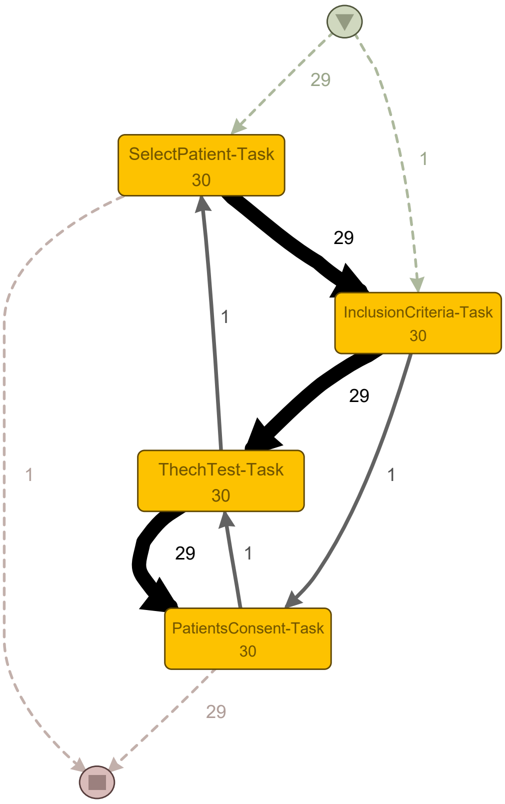


Figure A.26.: Case execution Tel Aviv CS2 Identification stage.

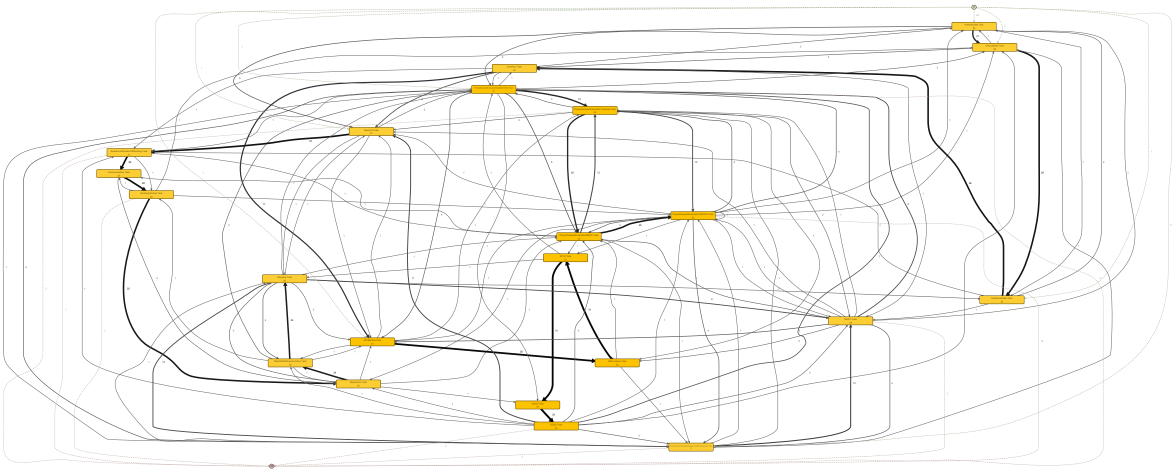


Figure A.27.: Case execution Tel Aviv CS2 Evaluation stage.

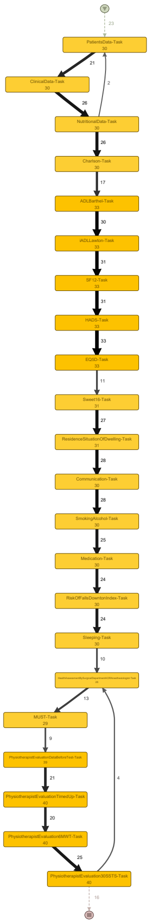


Figure A.28.: Case execution Tel Aviv CS2 Evaluation stage with abstracted paths.

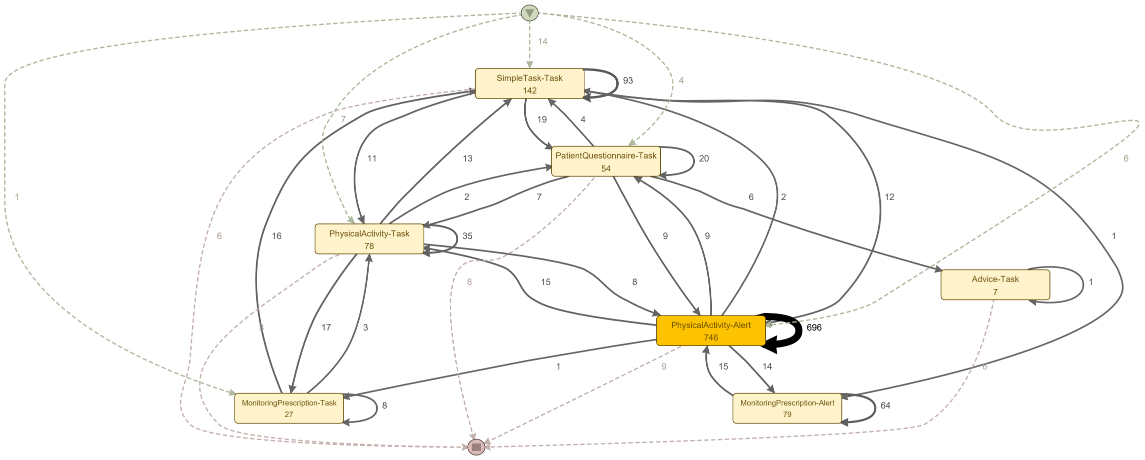


Figure A.29.: Case execution Tel Aviv CS2 Workplan stage.

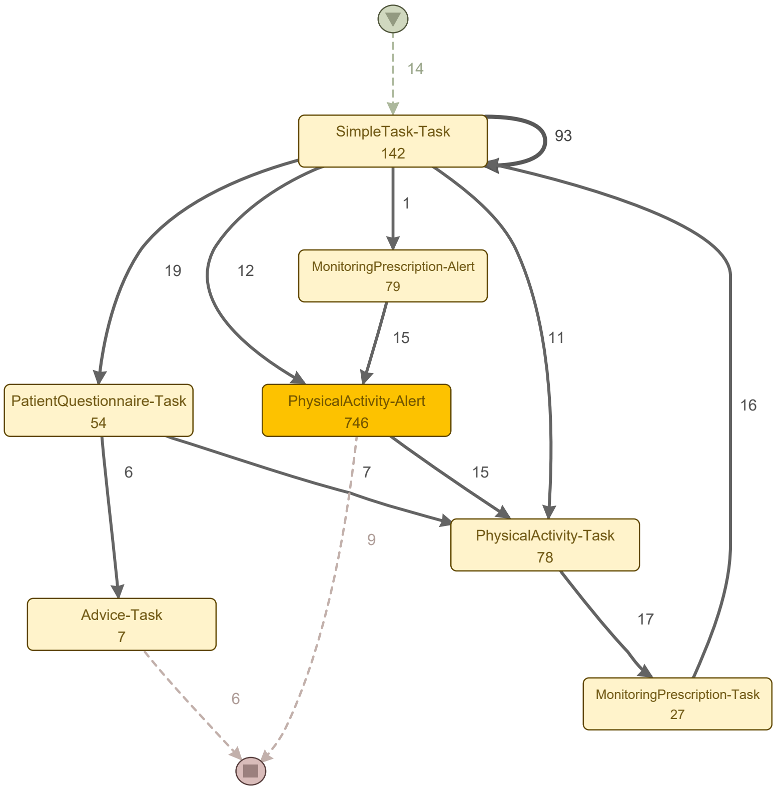


Figure A.30.: Case execution Tel Aviv CS2 Workplan stage with abstracted paths.

A.5.3. Case Execution Behavior in Lleida

	CS1					CS2					
Indicator	**Identification**	**Evaluation**	**Workplan**	**Discharge**	**Case Level**	**Identification**	**Evaluation**	**Workplan Pre**	**Workplan Post**	**Discharge**	**Case Level**
Number of cases	38	39	39	14	39	36	34	35	29	13	36
Number of activities	5	22	12	3	10	6	7	23	10	3	12
Number of overall paths	8	187	69	5	51	15	19	206	55	6	77
Mean number of paths per activity	1.6	8.5	5.8	1.7	5.1	2.5	2.7	9	5.5	2	6.4
Number of process variants	3	39	39	5	39	7	24	35	29	5	36
Maximum share of cases per variant	79%	3%	3%	29%	3%	58%	18%	3%	3%	46%	3%
Mean share of manual activated tasks	0%	15%	86%	0%	28%	0%	5%	67%	20%	0%	15%

Table A.5.: Case execution characteristics and run-time planning in Lleida.

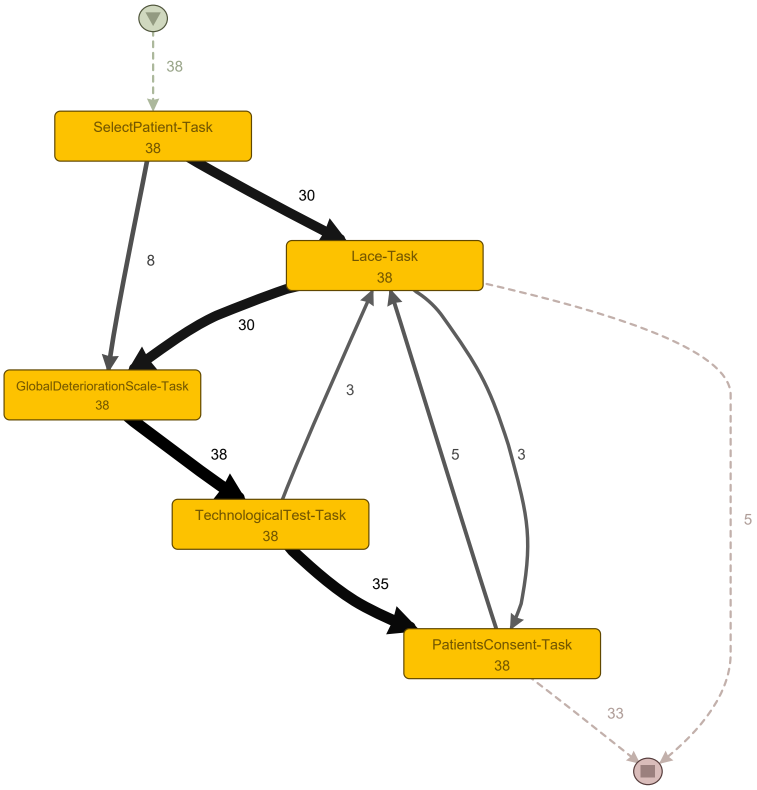


Figure A.31.: Case execution Lleida CS1 Identification stage.

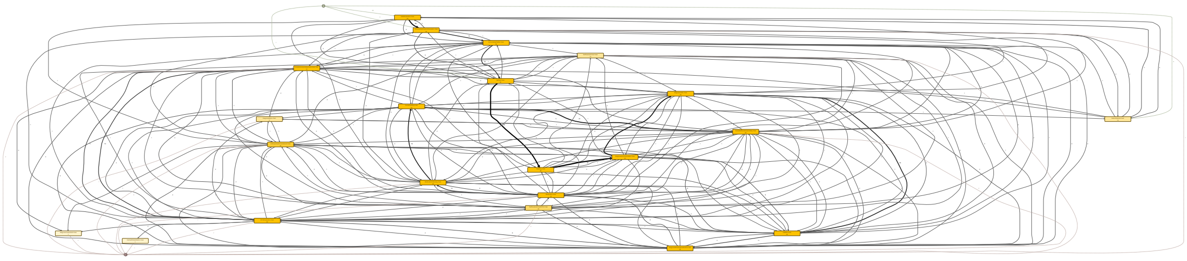


Figure A.32.: Case execution Lleida CS1 Evaluation stage.

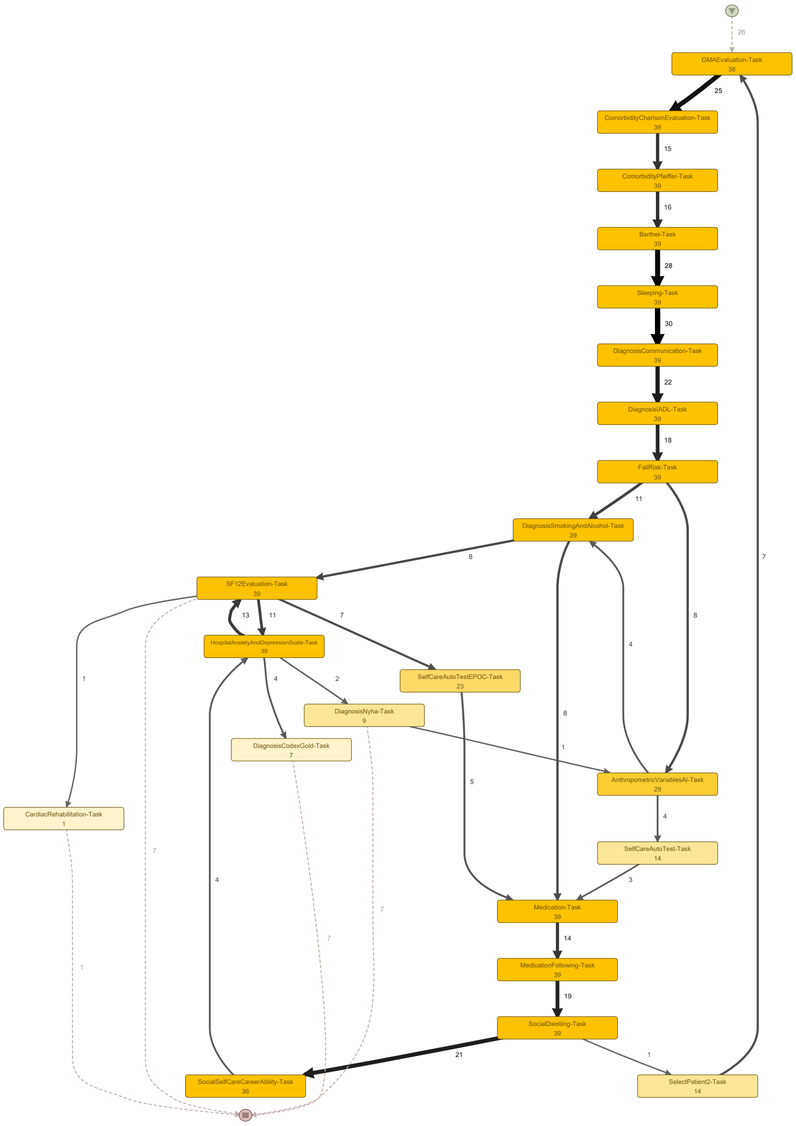


Figure A.33.: Case execution Lleida CS1 Evaluation stage with abstracted paths.

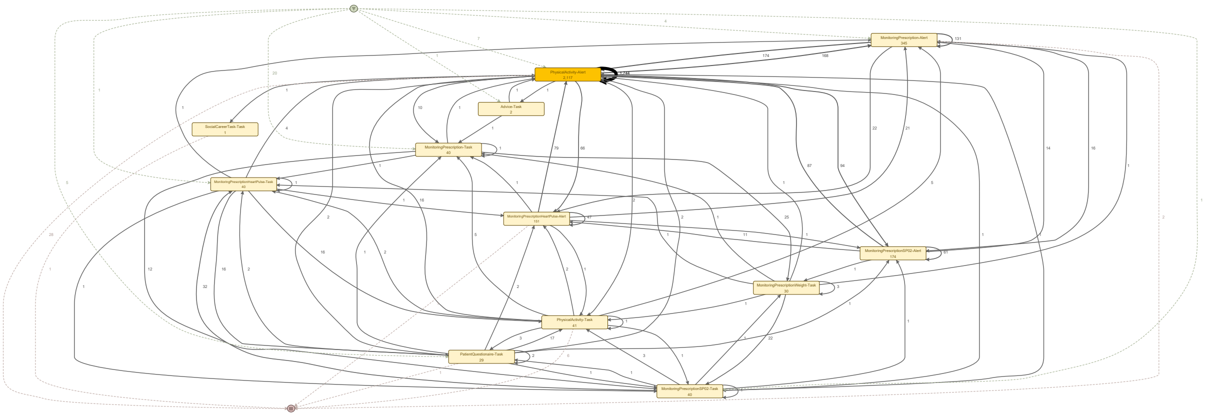


Figure A.34.: Case execution Lleida CS1 Workplan stage.

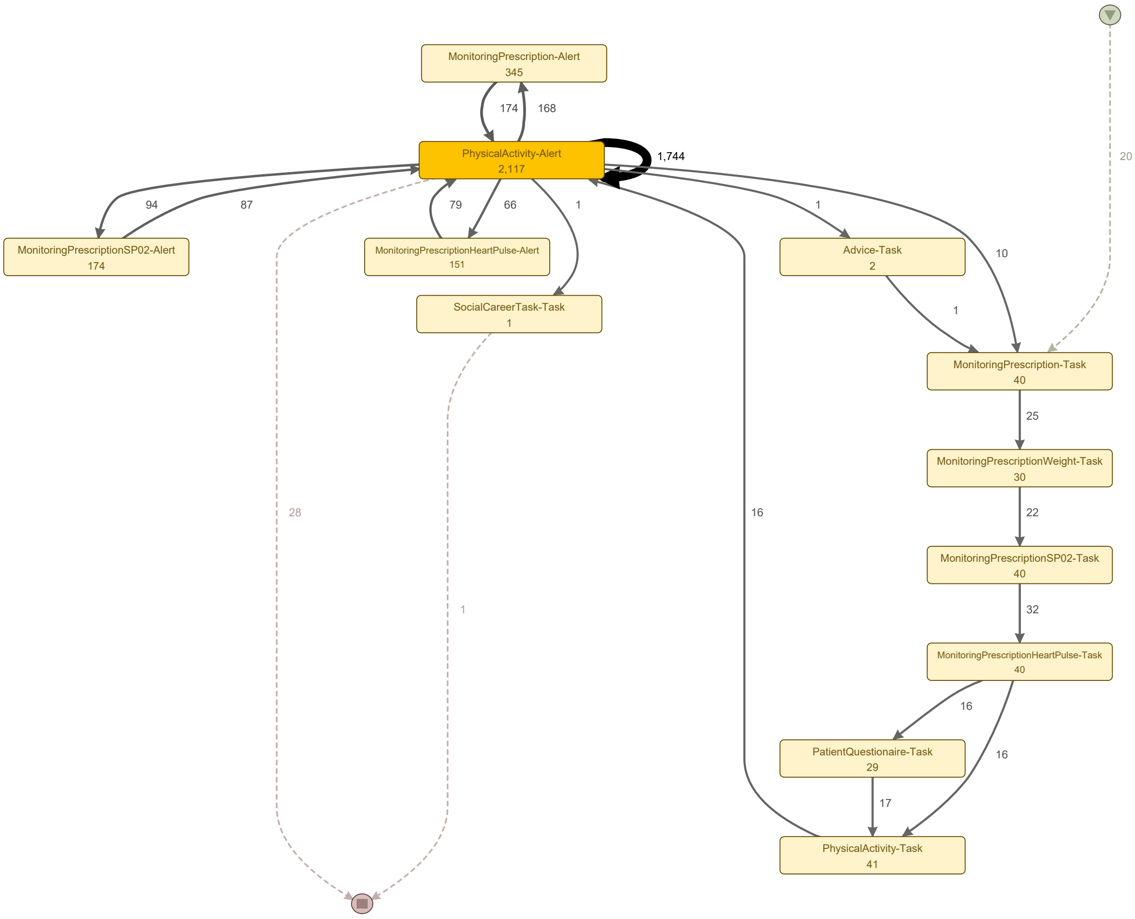

MonitoringPrescription-Alert
345
174
168
PhysicalActivity-Alert
2,117
1,744
20
94
87
79
66
1
1
MonitoringPrescriptionSP02-Alert
174
MonitoringPrescriptionHeartPulse-Alert
151
Advice-Task
2
10
SocialCareerTask-Task
1
1
MonitoringPrescription-Task
40
25
MonitoringPrescriptionWeight-Task
30
28
16
22
MonitoringPrescriptionSP02-Task
40
1
32
MonitoringPrescriptionHeartPulse-Task
40
16
16
PatientQuestionaire-Task
29
17
PhysicalActivity-Task
41

Figure A.35.: Case execution Lleida CS1 Workplan stage with abstracted paths.

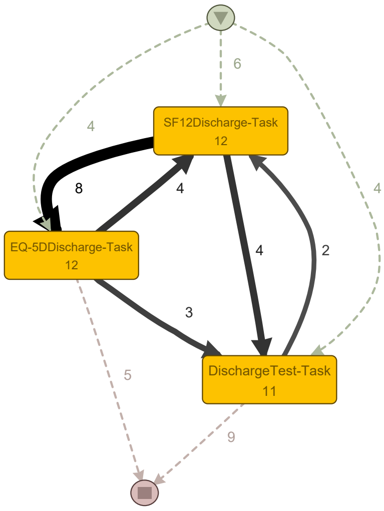


Figure A.36.: Case execution Lleida CS1 Discharge stage.

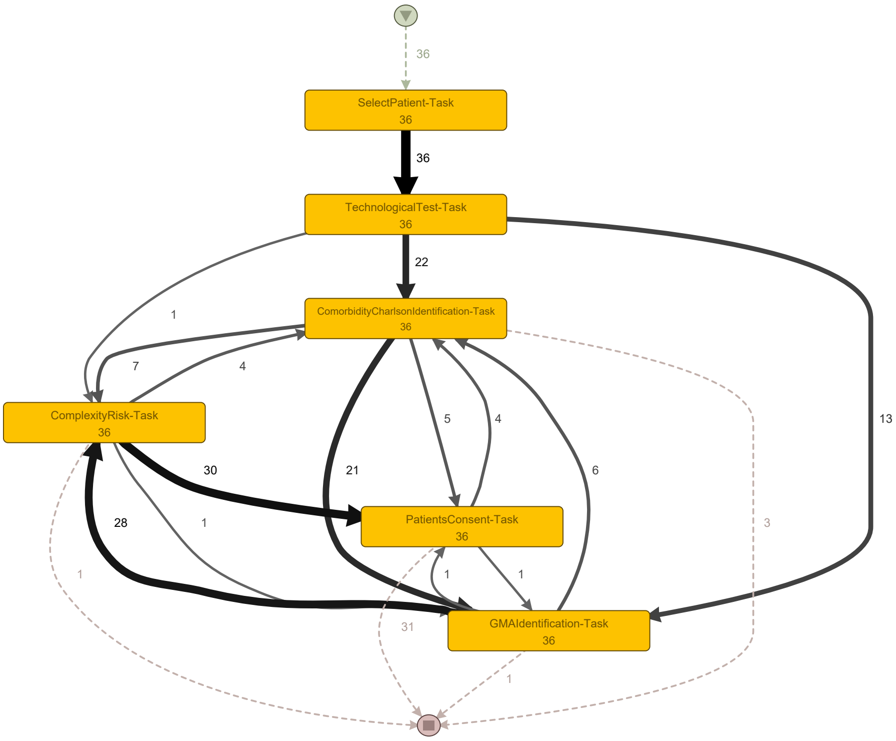


Figure A.37.: Case execution Lleida CS2 Identification stage.

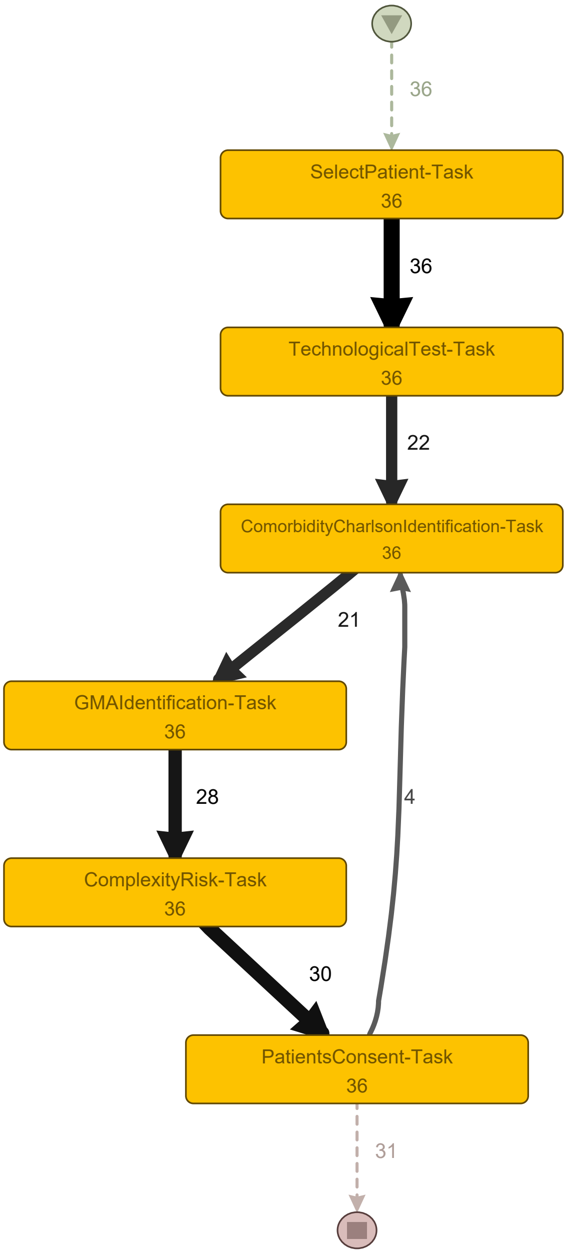


Figure A.38.: Case execution Lleida CS2 Identification stage with abstracted paths.

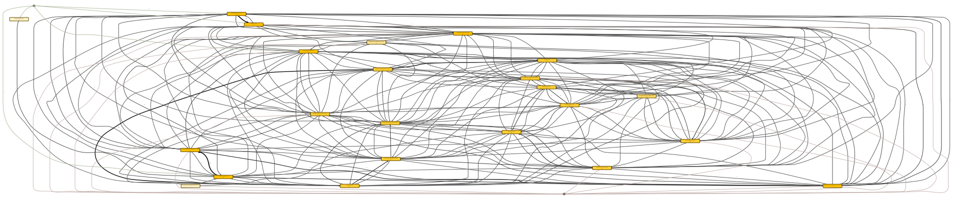


Figure A.39.: Case execution Lleida CS2 Evaluation stage.

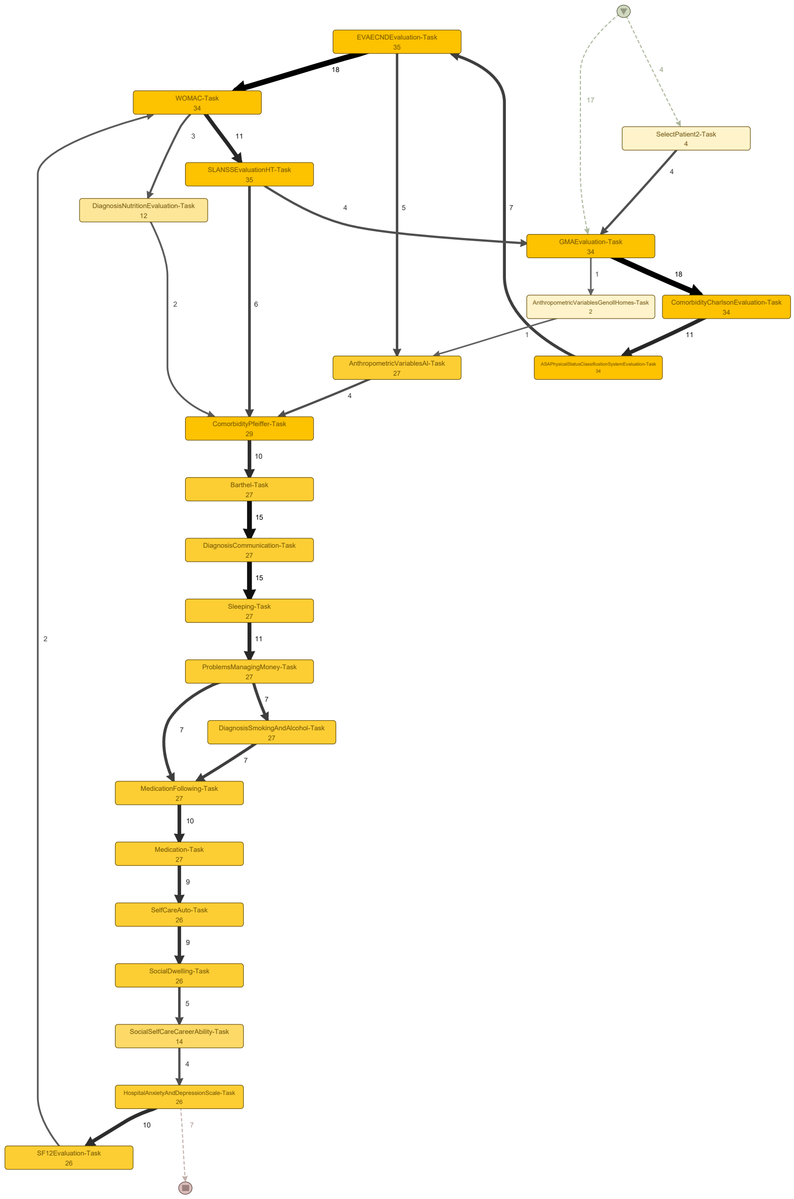

EVAECNDEvaluation-Task
35
18
WOMAC-Task
34
3
11
SLANSSEvaluationHT-Task
35
DiagnosisNutritionEvaluation-Task
12
17
4
SelectPatient2-Task
4
4
4
5
7
GMAEvaluation-Task
34
1
18
AnthropometricVariablesGenollHomes-Task
2
ComorbidityCharlsonEvaluation-Task
34
2
6
1
11
AnthropometricVariablesAI-Task
27
ASAPhysicalStatusClassificationSystemEvaluation-Task
34
4
ComorbidityPfeiffer-Task
29
10
Barthel-Task
27
15
DiagnosisCommunication-Task
27
15
Sleeping-Task
27
2
11
ProblemsManagingMoney-Task
27
7
7
DiagnosisSmokingAndAlcohol-Task
27
7
MedicationFollowing-Task
27
10
Medication-Task
27
9
SelfCareAuto-Task
26
9
SocialDwelling-Task
26
5
SocialSelfCareCareerAbility-Task
14
4
HospitalAnxietyAndDepressionScale-Task
26
10
7
SF12Evaluation-Task
26

Figure A.40.: Case execution Lleida CS2 Evaluation stage with abstracted paths.

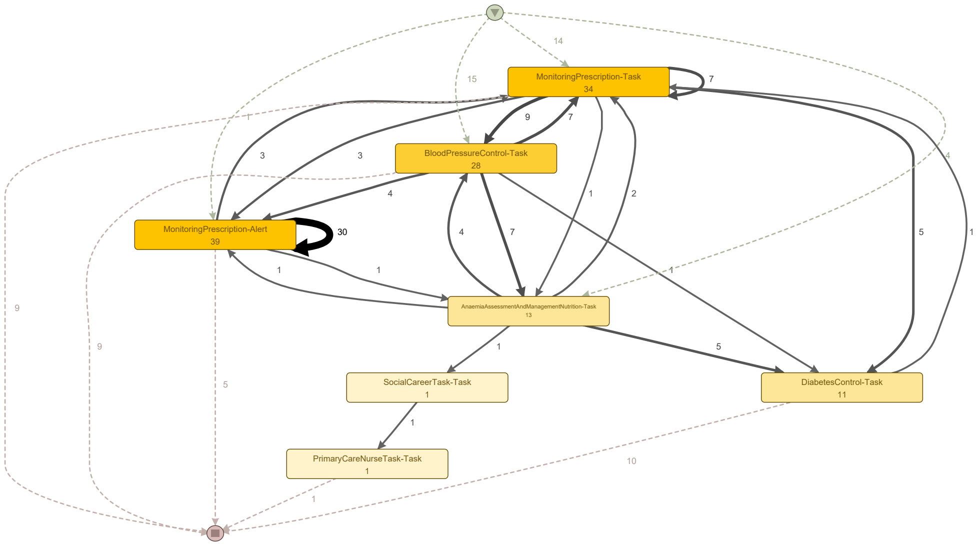

14
15
MonitoringPrescription-Task
34
7
9
7
3
3
1
BloodPressureControl-Task
28
4
1
2
4
MonitoringPrescription-Alert
39
30
4
7
5
1
1
1
1
9
AnaemiaAssessmentAndManagementNutrition-Task
13
9
1
5
5
SocialCareerTask-Task
1
DiabetesControl-Task
11
1
PrimaryCareNurseTask-Task
1
10
1

Figure A.41.: Case execution Lleida CS2 Workplan Pre stage.

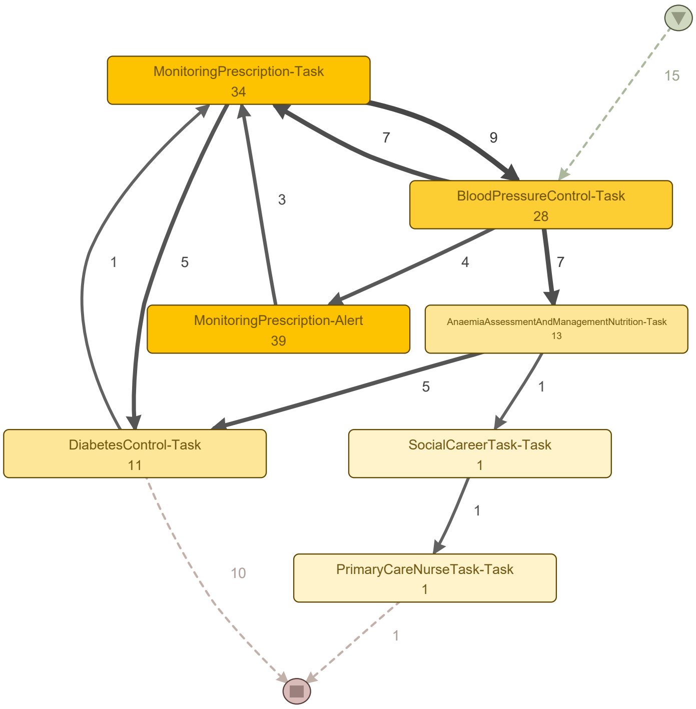

MonitoringPrescription-Task
34
15
7
9
3
BloodPressureControl-Task
28
1
5
4
7
MonitoringPrescription-Alert
39
AnaemiaAssessmentAndManagementNutrition-Task
13
5
1
DiabetesControl-Task
11
SocialCareerTask-Task
1
1
10
PrimaryCareNurseTask-Task
1
1

Figure A.42.: Case execution Lleida CS2 Workplan Pre stage with abstracted paths.

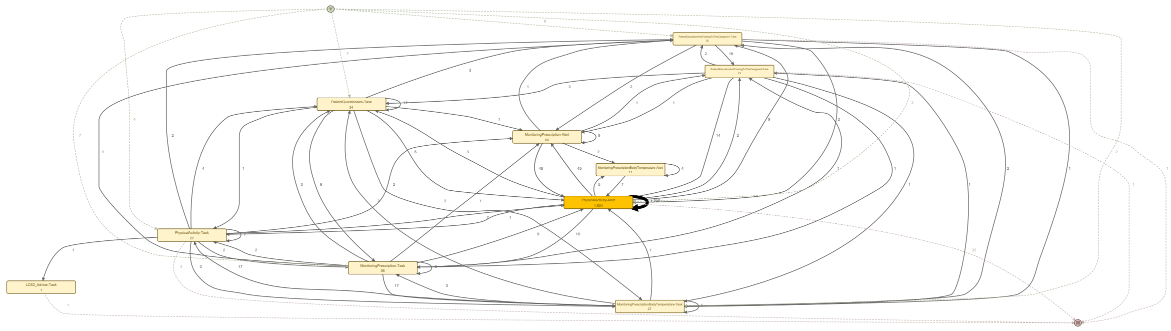


Figure A.43.: Case execution Lleida CS2 Workplan Post stage.

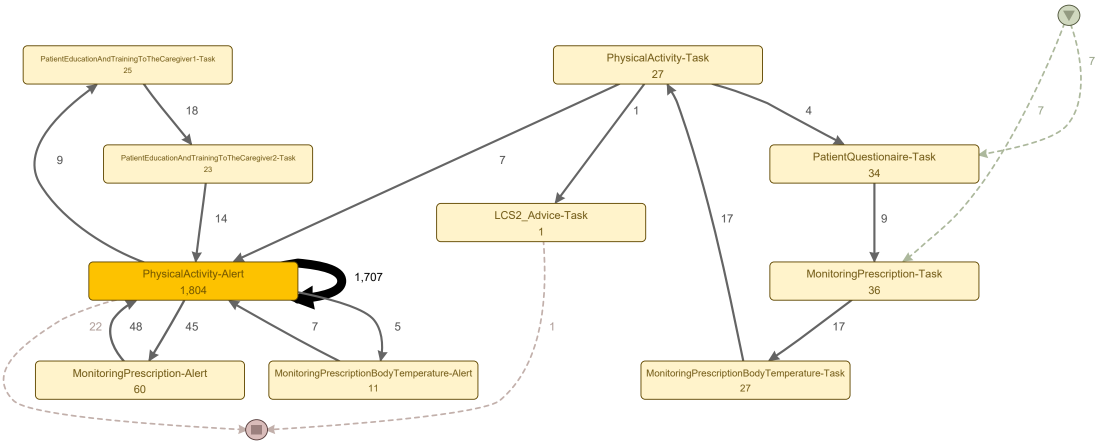

PatientEducationAndTrainingToTheCaregiver1-Task
25
PhysicalActivity-Task
27
18
9
PatientEducationAndTrainingToTheCaregiver2-Task
23
7
1
4
7
7
PatientQuestionaire-Task
34
LCS2_Advice-Task
1
14
17
9
PhysicalActivity-Alert
1,804
1,707
MonitoringPrescription-Task
36
22
48
45
7
5
1
17
MonitoringPrescription-Alert
60
MonitoringPrescriptionBodyTemperature-Alert
11
MonitoringPrescriptionBodyTemperature-Task
27

Figure A.44.: Case execution Lleida CS2 Workplan Post stage with abstracted paths.

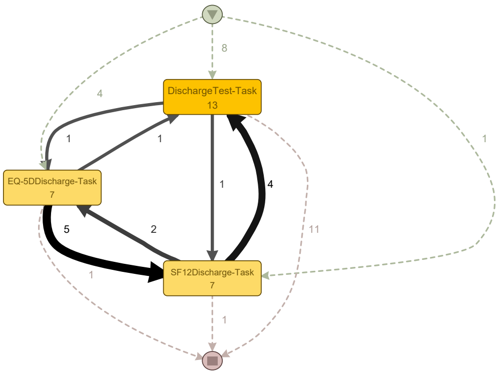


Figure A.45.: Case execution Lleida CS2 Discharge stage.