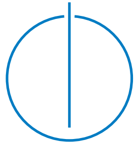
Martin Jergler

Distributed Workflow Coordination in Publish/Subscribe Environments

Technische
Universität
München



Technische Universität München

Fakultät für Informatik

Distributed Workflow Coordination in Publish/Subscribe Environments

Martin Jergler

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.

Vorsitzende(r): Prof. Dr. Nils Thürey

Prüfer der Dissertation:

1. Prof. Dr. Hans-Arno Jacobsen
2. Prof. Dr. Stefanie Rinderle-Ma,
   Universität Wien

Die Dissertation wurde am 29.05.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.09.2017 angenommen.

To my unborn kids.

Abstract

Workflow Management Systems (WFMSs) contribute towards the automation of business processes by reducing execution time, improving resource utilization, and enhancing service quality. The distribution of data together with legal regulations, the increasing demand for flexibility, and the massive scale of today's business processes, however, impose new challenges for WFMSs. Addressing them requires a paradigm shift in process modeling and corresponding system support. In this spirit, this work presents three approaches towards mitigating the described problems.

This entails the formalization of a foundation for the safe distribution and parallel execution of data-centric workflows over the publish/subscribe (pub/sub) abstraction to support geographical constraints on process data. We present a novel representation of data-centric workflows, modeled as Business Artifacts with Guard-Stage-Milestone (GSM) lifecycles, to exploit the loosely-coupled and distributed nature of pub/sub systems. GSM forms an abstraction of flexible business processes and we enable a workflow mapping by redefining key pub/sub constructs. As a result, once the workflow is mapped into the pub/sub abstraction, it inherits the loosely-coupled benefits of pub/sub while provably guaranteeing the original execution semantics.

In addition, we design a fully-distributed geo-scale WFMS to automatically execute geographically scattered GSM processes, while supporting locality of process- and data fragments. Our system is based on Workflow Units (WFUs), which form a unit of distribution, communicate over pub/sub, and manage individual process- or data fragments. We present two mapping implementations of GSM into WFUs: the baseline mapping (BLM) directly stemming from the pub/sub formalization, and the optimized context-aware mapping (CAM), which considers external event information to reduce the computational overhead and network communication. Our experiments show that both mappings are scalable but CAM outperforms BLM.

The horizontal scaling of WFMSs that are integrated over pub/sub requires replication of the workflow engine and instance dispatching, which comprises several pub/sub operations. To this end, we present an approach for multi-client transactions in distributed pub/sub systems. We formalize a pub/sub transaction as a sequence of pub/sub operations that are to be atomically processed but isolated from any

concurrent transaction, and where publications by one client can trigger further operations by other clients. Based on the a priori knowledge of a transaction coordinator (TXC), we present two implementations of our model: D-TX assumes no prior knowledge on operations providing sequential consistency and serializability. S-TX assumes full knowledge and manages isolation at the application level. Our experiments show that isolation and uncertainty about operations renders D-TX costly and suitable only for smaller configurations, in contrast to S-TX.

Zusammenfassung

Workflow Management Systeme (WFMS) tragen zur Automatisierung von Geschäftsprozessen bei, indem sie Ausführungszeiten reduzieren, die Ressourcenauslastung erhöhen, sowie die Servicequalität verbessern. Datenverteilung und daran gebundene gesetzliche Bestimmungen, steigende Flexibilitätsanforderungen und die enormen Ausmaße heutiger Geschäftsprozesse stellen jedoch neue Herausforderungen für WFMSs dar. Diesen entgegenzutreten erfordert einen Paradigmenwechsel in der Modellierung von Geschäftsprozessen und entsprechende Systemunterstützung. In diesem Zusammenhang stellt die vorliegende Arbeit drei Ansätze vor, die beschriebenen Probleme zu adressieren.

Zunächst wird die theoretische Grundlage formuliert, um datenzentrische Wokflows auf Basis der Publish/Subscribe (Pub/Sub) Kommunikationsabstraktion sicher verteilen und parallel ausführen zu können, und gleichzeitig geographische Rahmenbedingungen mitzuberücksichtigen. Um die lose Koppelung von Pub/Sub Systemen nutzen zu können, wird eine neuartige Darstellung datenzentrischer Workflows, modelliert als Geschäftsartefakte mit Guard-Stage-Milestone (GSM) Lebenszyklen, vorgestellt. GSM stellt eine Abstraktion flexibler Geschäftsprozesse dar; die Transformation eines Workflows wird dabei durch die Neuformulierung von Kern-Pub/Sub Konstrukten erreicht. Als Resultat erbt ein transformierter Workflow sämtlich Vorteile, die sich aus der losen Kopplung eines Pub/Sub Systems ergeben, und behält dennoch nachweislich seine ursprüngliche Ausführungssemantik.

Um geographisch verteilte GSM Prozesse automatisch auszuführen und gleichzeitig die Lokalität von Daten- und Prozessfragmenten sicherzustellen, wird ein vollständig verteiltes und geographisch skalierendes WFMS entwickelt. Das System basiert auf Workflow Units (WFUs), welche als Verteilungseinheit fungieren, mittels Pub/Sub kommunizieren und dabei einzelne Daten- und Prozessfragmente verwalten. Insgesamt werden zwei Abbildungen von GSM in die WFU-Repräsentation vorgestellt: Eine Baseline-Abbildung (BLM), welche direkt von der theoretischen Pub/Sub Formalisierung abstammt, und eine optimierte kontextbewusste Abbildung (CAM), welche Eventinformationen dazu benutzt, um den Berechnungsaufwand und die Netzwerkkommunikation zu reduzieren. Eine experimentelle Studie zeigt, dass beide Abbildungen skalieren, CAM jedoch BLM in jeder Hinsicht übertrifft.

Die horizontale Skalierung von Pub/Sub-basierten WFMSs erfordert eine Replizierung der Workflowengine und eine entsprechende Verteilung der Prozessinstanzen, was wiederum mehrere Pub/Sub Operationen umfasst. Aus diesem Grund wird ein Ansatz zur Transaktionsverwaltung in verteilten Pub/Sub Systemen vorgestellte, welcher Operationen von mehrere Nutzer unterstützt. Eine Pub/Sub Transaktion wird als Menge einzelner Pub/Sub Operationen definiert, welche atomar und isoliert von anderen Transaktionen ausgeführt werden sollen, wobei Events einzelner Nutzer Operationen anderer Nutzer auslösen können. Ausgehend von dem Vorwissen, das ein Transaktionskoordinator über die Transaktion besitzt, werden zwei Implementierungen des Modells vorgeschlagen: D-TX erwartet keinerlei Vorabwissen über einzelne Operationen und bietet sequenzielle Konsistenz sowie Serialisierbarkeit. S-TX erwartet vollständiges Transaktionswissen und verwaltet die Isolation auf Anwendungsebene. Eine experimentelle Studie verdeutlicht, dass die strikte Isolation und die Ungewissheit über Operationen D-TX, im Gegensatz zu S-TX, teuer und lediglich für kleinerer Systemkonfigurationen brauchbar machen.

Acknowledgments

All the work towards this dissertation and my doctoral degree took place at the Department of Informatics of the Technische Universät München under the supervision of Prof. Hans-Arno Jacobsen.

First, I want to express my sincere gratitude to my supervisor Prof. Hans-Arno Jacobsen for accepting me as the first Ph.D. student within his Alexander von Humboldt Professorship at TUM, for his continuous support, guidance, and encouragement throughout my research, and for his constant confidence in my person. He offered me great freedom in various aspects of my work and was always a reliable supporter.

I would also like to thank the rest of my thesis committee: Prof. Stefanie Rinderle-Ma from the University of Vienna for agreeing to be the second examiner and Prof. Nils Thuerey for acting as chair of the committee.

Huge thanks belong to my colleagues at the chair for their help, their inspiration, motivation, and for creating this wonderful atmosphere over the past years. Without you this work would not have been possible and I'm really happy about having earned some true friends. Thanks go to Mo Sadoghi for his input on the GSM mapping, and to Kaiwen Zhang for his discussions and ideas on the pub/sub transactions work. In particular, I would like to thank Matthias Kahl, Victor del Razo, Christoph Doblander, and the best officemate ever, Jose Rivera, for giving me so many moments of real quality time — not only during office hours but especially in our leisure time.

I also want to thank my students whose theses I was tutoring and whose work provided helpful input to my research. In particular, I want to thank Fabian Seebauer and Konstantin Bingmann for their work on the GSM editor, Manoj Mahabaleshwar for his work on the first GSM execution prototype, as well as Amit Sama and Gaganjot Singh for their work related to the pub/sub transaction approach.

Outside the academic world, I first and foremost want to thank my dearly belove wife Viviana for getting through this time with me and supporting me with all she could give. She never got tired of motivating me and still encouraged me when I was close to giving up — Vivi you changed my life and you mean the world to me.

Moreover, I want to express my deepest gratitude to my whole family for their moral support, their advice, and for simply being there whenever I needed them. I want to thank my parents Brigitta and Lorenz for their unconditional love, their patience, and their confidence in all of my decisions. I want to thank my brother Markus and my sister Johanna who calmed me down and encouraged me when things went not as intended. I also want to thank my “family in law” for welcoming me so warmly to their family and for being an important backing of my life.

Contents

Abstract **v**

Zusammenfassung **vii**

Acknowledgments **ix**

1 Introduction **1**

1.1 Motivation 4
1.1.1 Multi-organizational business processes 4
1.1.2 Flexible business processes 6
1.1.3 Data-centric workflows 7
1.1.4 Scalable workflow execution 9
1.2 Problem statement 11
1.3 Approach 14
1.3.1 Publish/Subscribe mapping of data-centric workflows 14
1.3.2 Geo-distribution of flexible business processes 15
1.3.3 Multi-client transactions in publish/subscribe 16
1.4 Contributions 17
1.5 Organization 19

2 Related Work **21**

2.1 Data-centric business artifacts 24
2.2 Distributed workflow execution 26
2.3 Transactions in message-oriented middleware 29

3 Background **33**

3.1 The Guard-Stage-Milestone meta-model . . . 33
3.1.1 Key constructs in GSM . . . 34
3.1.2 GSM operational semantics . . . 36
3.2 Distributed transactions . . . 40
3.2.1 Consistency models and consistent replication . . . 40
3.2.2 Isolation levels . . . 42
3.2.3 Distributed atomic commitment . . . 43

4 Publish/Subscribe Mapping of Data-centric Workflows 45
4.1 Formal data-centric workflow model . . . 46
4.1.1 Overview of GSM meta-model . . . 46
4.1.2 Example of data-centric workflow in GSM . . . 49
4.2 Publish/Subscribe schema . . . 52
4.3 Workflow mapping overview . . . 58
4.4 Mapping formalization . . . 60
4.4.1 Matching and notification policies . . . 60
4.4.2 Consumption policy . . . 69
4.5 Workflow mapping analysis . . . 71
4.5.1 Correctness . . . 71
4.5.2 Overhead in B-Step execution . . . 77
4.6 Foundation for distribution . . . 78
4.7 Summary . . . 83

5 Geo-Distribution of Flexible Business Processes 85
5.1 Flexible business processes formalism . . . 86
5.1.1 Case Management Model and Notation . . . 87
5.1.2 Guard-Stage-Milestones meta-model . . . 89
5.2 Geo-distributed data-centric workflow architecture . . . 92
5.3 GSM to Workflow Unit mappings . . . 95
5.3.1 Baseline mapping . . . 96
5.3.2 Context-aware mapping . . . 99
5.4 Geo-scale system deployment . . . 103
5.4.1 WFU Evaluation at WF Agents . . . 104
5.5 Experimental Evaluation . . . 106
5.6 Summary . . . 110

6 Multi-client Transactions in Distributed Publish/Subscribe 113
6.1 Distributed content-based publish/subscribe . . . 115
6.1.1 Event space formalization . . . 115
6.1.2 Elementary publish/subscribe operations . . . 116
6.1.3 Broker network . . . 117
6.2 Transactional content-based publish/subscribe model . . . 117
6.2.1 Definition and properties of a transaction . . . 118
6.2.2 Multi-user consistency . . . 119
6.2.3 Distributed isolation . . . 121
6.3 Dynamic Transaction Service . . . 122
6.3.1 D-TX overview . . . 123
6.3.2 D-TX client design . . . 124
6.3.3 D-TX broker design & protocol . . . 125
6.3.4 D-TX discussion . . . 132
6.4 Static Transaction Service . . . 133
6.4.1 S-TX overview . . . 133
6.4.2 S-TX client design . . . 134
6.4.3 S-TX broker design & protocol . . . 134
6.4.4 S-TX discussion . . . 138
6.5 Evaluation . . . 139
6.5.1 Overview . . . 139
6.5.2 Baseline implementation (BL-WAIT) . . . 140
6.5.3 Experiments . . . 140
6.5.4 Discussion . . . 146
6.6 Summary . . . 147

7 Conclusions 149
7.1 Summary . . . 149
7.2 Future work . . . 152

List of Figures 155

List of Tables 157

List of Algorithms 159

Bibliography **161**

Chapter 1

Introduction

Business Process Management (BPM) has emerged as a discipline that combines knowledge from information technology and management sciences, and applies it to operational business processes. The main intension of BPM is to understand such business processes and optimize them with respect to execution time, resource commitment, and service quality — in other words, to increase efficiency of the underlying business. Consequently, the scope of BPM in research and practice is rather broad and includes, among others, operations management, techniques for modeling, simulation, and analysis of business processes, as well as technologies for business process automation and workflow management [95].

A business process is a chain of events, activities, and decisions that happen within or among a set of organizations to generate value [25]. The structure of these elements and their relationship within a given process are used to classify the business processes [26]. For instance, in *tightly-structured* business processes, like in product manufacturing or sales, all possible activities and events, as well as their order can be determined a priori. This knowledge facilitates the formulation of decisions that have to be taken during process enactment and enables a precise description of the complete business process, which can be expressed in a model. A business process model represents at least a control-flow blueprint of the process, which every single process instance is supposed to follow (e.g., the disposition of a

particular product on an e-commerce platform). Well-established modeling languages for tightly-structured business processes are Petri nets, event-driven process chains (EPC), or the Business Process Model and Notation (BPMN) [72]. However, due to a lack of a priori process knowledge, not all processes can be described so rigorously.

Over the past decade, industrialized societies have faced a tipping point in the way work is carried out by people. The amount of routine work and, thereby, the number of tightly-structured processes like the ones mentioned above is declining compared to the amount of knowledge-intense work, which is ever increasing [93]. Knowledge work occurs in many domains such as health-care, insurance, science and engineering, project management, and governmental processes [61]. Here, business processes are *loosely-structured* and must support flexibility as often not all activities or events can be foreseen. We also refer to such processes as *flexible* business processes. A concrete set of activities and their order might differ from instance to instance depending on the current situation and the process context. Thus, decisions have to be made by human experts, a.k.a. knowledge workers, that are involved in the process. In this regard, the term *Case* has been coined to describe flexible and loosely-structured processes, and *Case Management* (CM) established as a new area of research within BPM. As one result, the Case Management Model and Notation standard (CMMN) [73] was recently published to provide adequate modeling support. To facilitate the situative character of flexible business processes, CMMN relies on a data-centric process formulation. The main advantage of data-centric models is that they include a data model to represent application-specific process information and support knowledge workers in their decision making (e.g., patient data in healthcare management). This allows for the flexible and fine-grained expression of control-flow decisions based on rules to evaluate the current data state of a process instance to evolve it to a subsequent state [23].

Business process models serve a twofold purpose: On the one hand, they provide a documentation and form the basis for discussion among business experts including process analysis, verification, and optimization. On the other hand, process models are frequently used to feed Process-aware Information Systems (PAIS)[95]. Common to PAIS is that they have an explicit process notion, know the process they support, and manage process data accordingly. Examples of PAIS are Enterprise Resource

Planning systems (ERP), Customer Relationship Management systems (CRM), or Workflow Management Systems (WFMS).

Workflows offer a technical perspective on business processes and enable their automated execution. A WFMS is configured with a model of the workflow; then, the WFMS controls the execution of a workflow instance by receiving events from the business environment and deciding which activities need to be executed next. Although, business process models and workflow models are quite similar, there is an important difference: workflow models must include sufficient information to process events and trigger activities, which are typically executed by external resources (e.g., knowledge-workers, web services, or actuator devices). This requires a technical description of possible event types and the communication endpoints of activities. Because of their event-based communication character with the business environment many WFMSs are realized using message-oriented middleware platforms [31]. Since WFMSs frequently need to manage huge amounts of concurrent and sometimes long-running instances, also distributed implementations have been considered for scalability reasons [28, 55].

In this work, we addresses the problem of workflow execution for supporting globally distributed and flexible business processes at scale. We investigate data-centric workflows, an abstraction of loosely-structured flexible business processes, which provide sufficient information for execution. Following the data-centric paradigm, we analyze strategies and techniques to distribute workflow execution using the publish/subscribe communication paradigm. Publish/subscribe enables decoupling of individual components, and provides scalable and reliable messaging. Our goals are twofold: on the one hand, we aim at providing locality and privacy of data, especially, in multi-organizational process settings, by distributing the complete workflow logic across a set of components. On the other hand, we want to scale workflow execution by distributing individual process instances across a set of components. For both directions, we investigate distributed content-based publish/subscribe middleware as integration platform to design, implement, and evaluate suitable solutions.

1.1 Motivation

Globalization of businesses, the technological advance, and the shift towards knowledge-intense work impose new challenges for workflow management in practice. These parameters are of particular relevance when considering the scale of today's business processes in terms of concurrent instances, the number of services to be coordinated, and the amount of data that needs to be managed in accordance with legal or organizational policies [34, 39, 93].

1.1.1 Multi-organizational business processes

The advance in information and communication technology is the main driver for many organizations to optimize their collaborative business processes with other organizations in order to unleash the potential for mutual benefits. Typically, workflows support these (globally) distributed business processes involving data and participants from disparate organizations and geographical locations. However, in such environments, for instance, in global corporations, business-relevant data is inherently distributed across data-centers and it is not uncommon that huge amounts of data need to be regularly moved across the globe, resulting in delayed decisions, decreased efficiency, and monetary loss.

Furthermore, compliance with legal regulations as the protection of business-relevant data, or other constraints that are imposed by individual organizations or even governments are hard to address. For example, the eighth Data Protection Principle of the Data Protection Act (DPA) in the United Kingdom requires that personal data (e.g., customer information) must not be transferred outside the European Economic Area unless the country or territory to which the data are to be transferred provides an adequate level of protection for personal data [29]. Similarly, the EU-U.S. Privacy Shield [27], successor of the Safe Harbor Agreement, requires that data transfers to third parties may only occur to such organizations that follow adequate data protection principles.

A domain that is subject to even stricter regulations on data exchange is healthcare and, in particular, the patient care sector. Usually, patient care involves many participants such as general practitioner (GP), hospitals, specialists, and recuperative care services, which need to coordinate their activities in order to treat a patient effectively. Often, these participants are independently organized businesses running their own processes and data management. Yet, the coordination of individual activities requires the referral of a patient from one organization to another including the exchange of relevant data (summarized as a patient record) [93]. This scenario is depicted in Figure 1.1.1. Patient data, however, is highly sensitive and must be processed were it is collected, unless a patient explicitly permits data exchange [8].

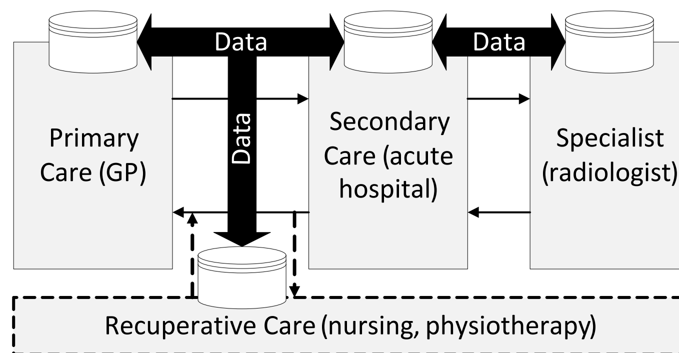


Figure 1.1.1: Referral chain in patient care business processes (adapted from [93]).

Despite their multi-organizational nature, it is not uncommon that a single global model is used to describe such processes. This global model is either agreed on before local processes are implemented at the participating organizations, or is imposed by the leading organization, which forces the other organizations to adapt their local processes. For instance, the supply chain management at Walmart defines a reference model that integrates the marketing processes and core data of their suppliers [99].

In summary, distributed processes require an adequate level of data protection to meet organizational and legal policies. At the same time, the amount of data that needs to be exchanged is a major hurdle in process execution.

1.1.2 Flexible business processes

The increase in knowledge-intense work, the acceleration of production cycles, and frequently changing business environments have re-shaped the character of business processes in various domains and demand for more flexibility in process modeling and execution.

In knowledge-intense processes, for instance, it is hard, and sometimes even impossible, to strictly model all necessary activities and their dependencies before actually executing the corresponding workflow. Traditional modeling techniques like BPMN [72] or BPEL [70] are inherently activity-centric and explicitly specify every possible control-flow decision. As a result, every single process instance is supposed to follow this rigid lifecycle. However, in flexible business processes, rarely two instances follow the same lifecycle and the lifcycle, in general, might be only loosely definable.

An example of a knowledge-intense and flexible business process occurs in healthcare [93]. A high-level overview of the typical patient care process is depicted in Figure 1.1.2. It is staged into five phases: *registration*, *assessment*, *treatment planing*, *treatment delivery*, and *review*. Two aspects are particularly noteworthy: First, each phase only represents a high-level summary of the associated activities. Especially, the stages *assessment*, *treatment planing*, and *treatment delivery* require expert knowledge and involve thousands of possible activities. Assessment, for instance, might include radiological, pathological, and all kinds of other methods. Only a physician with sufficient expertise is able to choose the appropriate methods and arrange them in a meaningful manner to generate a complete picture of the physical constitution of a particular patient. This decision process heavily depends on the patient itself, its history, and its symptoms—which, makes the process *ad-hoc*. In addition, it is hard to capture a complete set of all possible assessment methods, which requires ad-hoc adaptation of the set of available tools. Second, even the order of stages in this high-level example is hard to determine. In general, a patient can be discharged at any time. More interesting, however, is that the stages *assessment*, *treatment planing*, and *treatment delivery* can change from one to another at almost any time, always depending on the expert decisions of the medical personnel involved.

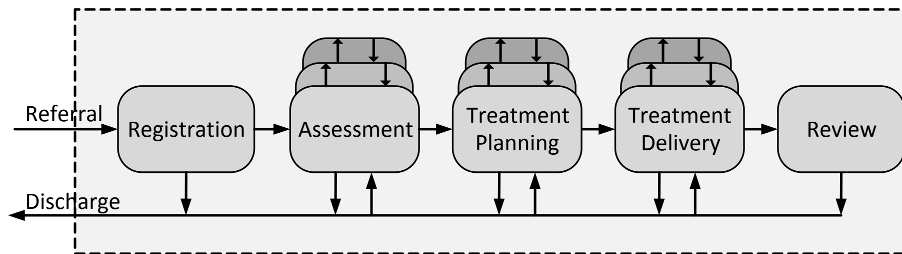


Figure 1.1.2: High-level overview of patient care business processes (adapted from [93]).

In summary, flexibility requires ad-hoc decision making and adaptation of the business process by domain experts at runtime, which activity-centric modeling and execution approaches do not support. For decision making, domain experts usually apply their knowledge to an individual case in a particular situation, e.g., the patient and its condition. Support systems and modeling approaches for flexible business processes therefore need to put a particular focus on the relevant information and include a corresponding model of the associated process data [39, 73, 23, 60, 62].

1.1.3 Data-centric workflows

In recent years, there has been a growing interest in frameworks for specifying and deploying workflows that combine both data and process as first-class citizens [2, 69, 79, 96, 23, 73, 52].

Data-centric workflows have a potential to address the problems described in Sections 1.1.1 and 1.1.2. Process and associated data are tightly-coupled in a sense that both are expressed in a single model without giving explicit favor to one of them. This simplifies workflow distribution according to geographical, organizational, and legal constraints as only a single model needs to be distributed. In addition, flexibility is supported as process-relevant data is part of the process model. The control-flow is modeled as declarative rules over the data model, which facilitates ad-hoc decision making.

In this work, we consider one such data-centric BPM approach called *Business Artifacts* (BA) [14, 20, 69] and a recent meta-model for modeling business artifacts

called *Guard-Stage-Milestone* (GSM) [37, 23, 38]. In the artifact-centric paradigm, business processes are modeled as interactions of key business-relevant, conceptual entities called *Business Artifacts* (or "artifacts", for short). Artifacts are modeled using an *information model* that includes attributes for storing all business-relevant information about the artifact, and a *lifecycle model* that represents possible ways the artifact might evolve over time. The artifact approach typically yields a high-level factoring of business processes into a handful of interacting artifact types.

The recently introduced data-centric workflow model known as *Business Artifacts with Guard-Stage-Milestone Lifecycles* meta-model [37, 38, 23] provides a declarative approach for specifying artifact lifecycles. GSM supports parallelism and modularity, with an operational semantics based on a variant of Event-Condition-Action (ECA) rules. There are four key elements in the GSM meta-model:

(a) The *information model*, which captures all relevant process data including application data as well as status information about the process itself.

(b) *Milestones*, which correspond to business-relevant operational objectives that are achieved (and possibly invalidated) based on triggering events and/or conditions over the information models of artifact instances.

(c) *Stages*, which correspond to clusters of activity intended to achieve milestones.

(d) *Guards*, which control when stages are opened or closed, respectively.

Multiple stages of an artifact instance may be active at the same time, which enables parallelism. Hierarchical structuring of the stages supports a rich form of modularity.

The operational semantics of GSM is characterized by how a single *event* from the business environment is incorporated into the current *snapshot* of the information model of a GSM-based system [23]. This semantics extends the well-known Event-Condition-Action (ECA) rule paradigm. It is centered around business steps (or B-steps, for short) that focus on the full impact of incorporating incoming external events. In particular, the focus is on what milestones (i.e., goals or objectives) are achieved or invalidated and what stages (i.e., activities) are opened and closed, as a

result of the incoming event. Changes in milestone and stage status are treated as internal *status events* and can trigger further status changes in the B-step. Intuitively, a B-step corresponds to the smallest unit of business-relevant change that can occur to a data-centric workflow.

In this thesis, we rely on the incremental operational semantics introduced in [23], which resembles the incremental application of ECA-like rules providing a natural and direct approach for its implementation.

1.1.4 Scalable workflow execution

Workflow Management Systems (WFMS) are software systems, which are integrated in an enterprise infrastructure or provided as a cloud service to enable automatic workflow execution. A model of the workflow, either explicitly created by a domain expert or derived from a corresponding business process, is passed to the WFMS, which controls the execution of individual workflow instances.

A single instance of the workflow represents the concrete execution of all relevant activities necessary to process the workflow for a given case or scenario. This can be, for example, all treatments delivered to patient X who reported about constant headache at his GP, or all steps involved in the disposal and delivery of a new iPhone 7 to customer Y at an online retailer. Workflow execution includes the invocation of the individual activities and their mutual data exchanges. The communication with the business environment is based on events received from or delivered to process participants like human beings, e.g., customers and patients, or IT support systems, e.g., ERP systems, databases, actuator devices, web services, etc.

WFMSs typically comprise numerous components, reveal plenty of dependencies, and show complex interaction patterns [55]. In addition, such systems are increasingly dynamic and require adaptations or elastic provisioning [49]. Hence, components must be added, removed, or adjusted ad-hoc and without disrupting the execution. Middleware services are used in this context as a coordination mechanism for individual components [9, 77]. On the one hand, a middleware should support

non-functional requirements like scalability and availability, on the other hand, it must allow to express the required degree of coordination—which is often a trade-off.

An increasing amount of WFMSs are realized in a distributed fashion using event-based coordination [45, 46, 49, 55]. Some WFMSs, for instance, consist of multiple components for data access and control flow computations [45]. The execution of a workflow requires these components to coordinate according to a protocol that depends on the atomicity and consistent order of a set of operations [46, 55].

In addition, a single workflow instance often involves significant communication with its environment and it is not uncommon that a WFMS needs to handle thousands of instances at a time; consider, for instance, the number of concurrent users and instances on sales platforms like Salesforce or Amazon [7]. To provide an elastically scaling service, industry-strength WFMSs assign individual instances to workflow agents [49]. Each agent is a replica of the WFMS dedicated to handle a single instance. A load balancer notices new instances and dispatches them across available agents for further processing. Again, the dispatching procedure requires a set of operations to be executed atomically and consistent with a protocol specific order. In general, this way of load balancing can be considered a design pattern for many PaaS cloud services (e.g., using Docker containers) [49, 90].

In recent years, publish/subscribe (pub/sub) has emerged as a popular middleware for coordinating enterprise systems with complex interaction patterns [17, 41]. Individual components (a.k.a. *clients*) *publish* data to the pub/sub service, which is delivered to a set of matching consumers that have previously *subscribed* their interest. Pub/sub systems exhibit strong decoupling properties across clients, which simplifies application development and allows for dynamic interactions. However, these decoupling properties limit the ability for clients to coordinate, since each pub/sub operation is processed independently and asynchronously but only limited guarantees are given for operation groupings.

Large-scale applications often employ distributed pub/sub to improve scalability [41]. In this thesis, we study the distributed content-based pub/sub system model [54]. An overlay network of brokers forwards subscriptions and publications according to their content. Each broker performs matching and routing functions to disseminate

publications and subscriptions to intended recipients. Specifically, each broker maintains a Subscription Routing Table (`SRT`) to propagate subscriptions to potential publishers and a Publication Routing Table (`PRT`) to store all known subscriptions. Incoming publications are matched against the `PRT` and forwarded to the next hops in the network until they reach the matching clients.

However, supporting the above scenarios requires the transactional grouping of various operations, which is not possible today.

1.2 Problem statement

Workflow Management Systems can support the automation of business processes and optimize the efficiency of the underlying business. However, the efficient automation of flexible business processes that involve multiple organizations at increasing scale and in accordance with legal or organizational regulations is challenging. Suitable solutions require to rethink how business processes are traditionally modeled and executed. In this context, publish/subscribe can provide a flexible, scalable, and available coordination platform for application and component integration. This work concentrates on three main research objectives:

1. *Partition data-centric workflows into data-access and control-flow components at a granularity, which is suitable to meet the data management requirements imposed by individual organizations or legislative authorities, and provide a reasoning about their effective distribution based on publish/subscribe coordination.*
2. *Design a distributed workflow management architecture that leverages the advantages of publish/subscribe middleware for deploying and executing data-centric workflows at geo-scale, while providing data locality for efficiency and policy compliance.*
3. *Define a transaction concept for publish/subscribe and provide suitable techniques for distributed publish/subscribe systems to support the horizontal scaling of workflow management systems build on top of such platforms.*

Although, the number of flexible business processes is increasing, still, the majority of current workflow management systems focuses on tightly-structured processes modeled in procedural languages. Data-centric models, to the contrary, unify process and data perspectives and offer the potential to overcome this limitation [16]. Data-centric approaches focus on the process goal and provide evolution based on the current system state by using declarative rules instead of a rigid control-flow.

At the same time, the vast majority of workflow management systems is designed to support only processes within a single organization. Such systems are either centralized in nature, relying on centralized processing of associated data, or support only restricted forms of distributed execution without considering data appropriately [12, 24, 16]. Since multi-organizational scenarios typically reveal an inherent distribution of the relevant information across multiple data centers, such designs complicate global workflow optimization and the adherence to data protection policies.

The support for flexible, multi-organizational business processes at geo-scale, while respecting locality of process and data, requires fully-distributed approaches to manage data accordingly. The state of the art are web services-based architectures, which, however, are either modeled with procedural languages [72, 70], or in the case of data-centric specifications, use a centralized engine to manage data and coordinate services [15, 31, 58, 68].

Providing distributed execution support for data-centric workflows is challenging for several reasons. Provision of data locality to meet compliance with data protection requires a concept to distribute data access, rule evaluation, and process control for data-centric workflows over multiple process components. The integration platform for these process components must be scalable, reliable, and flexible to enable process changes such as the addition of new components. For performance reasons, the number of affected process components and the number of messages exchanged during process evolution should be minimized.

In summary, the ultimate goal of distributing a data-centric workflow is to achieve an effective grouping of workflow components while respecting a set of constraints such as the infrastructure topology, geographical constraints, or pricing factors, while minimizing communication or data transport costs.

In addition, the integration of large-scale applications and components in enterprise systems based on publish/subscribe middleware requires the grouping of various operations including transactional properties like atomicity, consistency, isolation, and durability, summarized as *ACID* properties.

To the best of our knowledge, there exists no definition of ACID semantics in the context of pub/sub. Adapting the ACID properties from databases to pub/sub is challenging because both types of systems fundamentally differ in their interaction paradigm, operation sets, and processing model. Yet, a precise formulation of the ACID properties is crucial to reason about an execution model for multi-client pub/sub transactions.

Furthermore, distributed pub/sub systems introduce a high degree of concurrency in managing the state of the various brokers. In particular, modeling consistency and isolation is non-trivial because the main focus lies on synchronizing routing tables across multiple brokers, which are not simply just replicated because not every pub/sub operation is transmitted to every broker.

Moreover, it is challenging for pub/sub users, which are fundamentally decoupled in nature, to be able to express a working order of operations within the context of a single transaction. In database systems, this is normally not an issue since each transaction either involves only a single client, or clients which are able to directly coordinate with one another. In some cases, there is a need to be able to express an order of operations within a transaction in an ad-hoc fashion, without relying on the clients having prior knowledge of all involved parties. In other cases, the challenge is to identify viable system assumptions, such as which a priori knowledge can be assumed for a transaction in given scenarios. Based on these assumptions, algorithms need to be developed to provide an efficient and scalable integration of the model into a distributed pub/sub service.

1.3 Approach

In summary, the objectives of this work are threefold. First, developing a fragmentation of data-centric workflows into sets of data-access and control-flow components by mapping the workflow into corresponding publish/subscribe primitives with equivalent execution semantics and providing the foundation for an effective distribution compliant with given constraints. Second, designing a geo-scale distributed workflow management architecture for executing flexible data-centric business processes, while providing data-locality for efficiency and policy compliance. Third, defining transactional semantics for publish/subscribe and providing suitable system support for distributed pub/sub implementations to support the horizontal scaling of workflow management systems built on top. We now briefly introduce these approaches.

1.3.1 Publish/Subscribe mapping of data-centric workflows

In this approach, we focus on how data-centric workflows specified in GSM can be fragmented into sets of multiple data-access and control-flow components by employing the publish/subscribe (pub/sub) abstraction. We believe that the loosely-coupled nature of pub/sub systems provides a convenient substrate for workflow execution: adaptations like the addition or removal of individual components can be accomplished during runtime by (un-)subscribing to events that drive the execution. The decoupling of individual workflow components facilitates their ability for migration and enables effective scalability of the system.

Starting with an information model and a set of data-centric workflow primitives (based on a set of acyclic ECA-style rules) that rely on an incremental operational semantics, we develop a complete mapping of data-centric workflows into the pub/sub abstraction. We enable this workflow transformation by redefining and formalizing key pub/sub constructs such as subscriptions and publications together with their matching conditions, as well as consumption and notification policies. As a result, once a data-centric workflow is transformed into the pub/sub abstraction, it seamlessly inherits the distributed and loosely-coupled benefits of pub/sub.

After laying the foundation for mapping data-centric workflow primitives to publish/subscribe primitives, we proof the maintenance of the operational semantics and quantify the execution cost in terms of messages. This foundation can now be applied to identify an optimal workflow distribution that conforms to given constraints. We formalize this distribution problem as the colored multiway cut problem and show its intractability by reduction of the multiway cut problem to our distribution problem.

1.3.2 Geo-distribution of flexible business processes

In this approach, we present a geo-distributed execution architecture for flexible data-centric business processes in multi-organizational scenarios based on GSM [23]. We show that GSM forms a generalization of flexible business processes and the execution semantics of CMMN [73].

The core of our system is a distributed workflow engine for GSM that supports locality of data in such a way that system components accessing particular data, are deployed in the IT infrastructure of the data owner at geo-scale. The theoretical foundation for this system is set in Chapter 4, where we described a publish/subscribe formulation of GSM. The chosen loosely-coupled nature of pub/sub supports both the ad-hoc character of flexible business processes as well as the rule-based execution semantics. In contrast to the formalization, this approach provides a geo-scale system architecture including an implementation, optimization, and experimental evaluation.

We introduce *Workflow Units* (WFUs) as distributable system components that communicate over pub/sub, and manage individual attributes from the data model or execute workflow rules. A WFU subscribes to the relevant attribute changes for rule evaluation or attribute access, performs the rule evaluation or data access in the context of an event from the environment, and publishes results to other interested WFUs. The WFU representation of the workflow partitions the global data model and control flow into fragments. This fragmentation is the basis for a location-aware deployment of the business process, where each WFU is deployed in the IT infrastructure of the organization that has the rights to manage the data accessed by the WFU.

1.3.3 Multi-client transactions in publish/subscribe

The foundation of our multi-client transactions approach is a formal publish/subscribe (pub/sub) model, in which we define all pub/sub operations as filters in a global event space. A transaction is modeled as a sequence of these operations. We also assume that certain operations can trigger other operations, e.g., a publication can trigger a subscription at the receiving client. The ACID semantics are then formalized over a sequence of operations. Consistency is defined as *sequential consistency* imposing a total-order relation on all operations in the sequence. Isolation is defined as *serializability*, which means that the result of executing two transactions concurrently should be the same as if both were executed one after another. In our implementation, a transaction is always managed by a transaction coordinator (TXC). We distinguish two different approaches based on the a priori knowledge of the TXC.

In S-TX, we assume that the TXC has static knowledge about all operations in the transaction and transactions are isolated at the application level. Ordering is realized by attaching a dependency list to every operation, referencing prior operations; brokers check if all dependencies are fulfilled before processing an operation. The routing states, `SRT` and `PRT`, are represented as *conflict free replicated datatypes* (CRDTs) [59]; hence, updating the routing state is either appending or removing from the CRDT. For atomicity, we introduce a special commit operation that contains a dependency to the last operation of the transaction.

In D-TX, the operation set is assumed to evolve at runtime (dynamic), i.e., the TXC does not know operations issued by other clients. In addition, no assumptions are made regarding isolation of concurrent transactions. Instead of dependencies, an acknowledgment mechanism ensures that a certain operation is completed, i.e., fully propagated through the system, before the next operation is processed. Serializability is realized by adopting a snapshot isolation algorithm, which first takes a globally consistent snapshot of the brokers' routing state. This snapshot is used to process all operations of a transaction. When committing the transaction, the snapshot is analyzed for conflicts with concurrent transactions and, either merged with the stable routing state (commit) or discarded in case of conflicts (abort). For this purpose, we adapt the well-known *2-phase commit* algorithm for atomic commitment to pub/sub.

1.4 Contributions

The main contributions from our formal publish/subscribe mapping of data-centric workflows into the publish/subscribe abstraction are:

i. We provide a formal model of data-centric workflows based on the Guard-Stage-Milestone (GSM) metamodel together with a suitable formalization of the publish/subscribe abstraction.

ii. We present a fragmentation of data-centric workflows into a set of data-access and control-flow components by defining a mapping of core workflow constructs into our publish/subscribe formalization to provide the foundation for the distributed and parallel execution.

iii. We provide a detailed theoretical analysis of our mapping. We proof that a publish/subscribe representation resulting from our mapping maintains the execution semantics of the original data-centric workflow model. Also, we quantify the overhead of workflow execution under the publish/subscribe formulation in terms of required messages.

iv. We analyze the complexity of the optimal workflow distribution by formalizing the problem as the colored multiway cut problem and show its intractability through reducing the multiway cut problem to this distribution problem.

v. We provide a greedy algorithm with a constant factor approximation for the workflow distribution problem.

The main contributions from our geo-distributed execution approach for flexible business processes are:

i. We analyze formalisms to model flexible business processes based on the case management standard (CMMN) and the Guard-Stage-Milstone metamodel (GSM) and show that CMMN models can be faithfully expressed by GSM.

ii. We describe a geo-scale execution architecture for GSM workflows based on Workflow Units (WFUs) residing on top of a distributed publish/subscribe middleware service.

iii. We present two mappings of GSM into WFUs: the baseline mapping (BLM) directly stems from our theoretical publish/subscribe formulation of data-centric workflows. The optimized context-aware mapping (CAM) considers knowledge about external event types to reduce the number of WFUs involved and the number of messages generated in the context of a single execution step.

iv. We provide an extensive experimental study evaluating and comparing BLM and CAM with respect to process latency and throughput. We show that CAM offers better performance, especially, for sequential task patterns.

The main contributions from our multi-client transaction approach for distributed publish/subscribe systems are:

i. We present a formal model for supporting transactions using publish/subscribe operations. We propose different levels of the ACID semantics for expressing multi-user transactions with varying guarantees and requirements with respect to a priori knowledge.

ii. We propose D-TX, our first solution for supporting transactions in the context of a distributed content-based publish/subscribe system. D-TX allows a set of operations to be defined at run-time, provides sequential consistency, serializability, and atomicity.

iii. We propose S-TX, our second distributed solution, which relies on static knowledge of all operations included in a transaction, provides weak isolation (application level), and sequential consistency (using *conflict-free replicated datatypes*), and atomicity.

iv. We provide implementations of S-TX and D-TX in a distributed pub/sub system, together with a comprehensive evaluation that compares the strengths of both solutions with a baseline solution, which mimics part of the transaction behavior by introducing manual operation delays.

Parts of the content and contributions of this work have been published in or are submitted to the following venues:

- M. Sadoghi, M. Jergler, H.-A. Jacobsen, R. Hull, and R. Vaculín. Safe Distribution and Parallel Execution of Data-Centric Workflows over the Publish/Subscribe Abstraction. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(10):2824–2838, 2015

- M. Jergler, M. Sadoghi, and H.-A. Jacobsen. D2WORM: A Management Infrastructure for Distributed Data-centric Workflows. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1427–1432, 2015

- M. Jergler, H.-A. Jacobsen, M. Sadoghi, R. Hull, and R. Vaculín. Safe Distribution and Parallel Execution of Data-centric Workflows over the Publish/Subscribe Abstraction. In *32nd IEEE International Conference on Data Engineering (ICDE)*, pages 1498–1499, 2016

- M. Jergler, M. Sadoghi, and H.-A. Jacobsen. Geo-Distribution of Flexible Business Processes over Publish/Subscribe Paradigm. In *Proceedings of the 17th ACM International Middleware Conference*, pages 15:1–15:13, 2016

- M. Jergler, K. Zhang, and H.-A. Jacobsen. Multi-client Transactions in Distributed Publish/Subscribe Systems. *Submitted to 18th ACM International Middleware Conference*, 2017

1.5 Organization

The rest of this thesis is organized as follows. Chapter 2 presents related work in the area of workflow management with particular emphasis on data-centric modeling, distributed approaches, the execution of data-centric workflows, and transactions in publish/subscribe systems. Chapter 3 provides background information on the Guard-Stage-Milestone meta model and distributed transactions.

Chapters 4 and 5 elaborate on the mapping of data-centric workflows into the publish/subscribe abstraction and the corresponding geo-distributed execution architecture.

Chapter 4 first presents formalizations for both data-centric workflows and the publish/subscribe paradigm. Then, it gives a detailed account of the mapping functions between both abstractions, putting a particular focus on the subscription component. Next, it provides an analysis of the mapping by proofing its correctness and estimating the execution costs. Finally, it presents the foundation for effective workflow distribution together with a complexity analysis.

Chapter 5 presents the geo-distributed execution architecture for flexible business processes. First, it presents CMMN and GSM as abstractions of flexible business processes and describes their relationship. Then, it provides an overview of the distributed execution architecture based on Workflow Units (WFU). Next, it describes BLM and CAM, two different mappings of GSM into WFUs. Finally, it provides details about workflow deployment and a discussion on the results collected from an experimental study.

Chapter 6 covers multi-client transactions in distributed pub/sub systems. First, it introduces a use-case scenario by modeling the instance dispatching problem from workflow management as pub/sub interactions. Then, it provides a formal transaction model for pub/sub operations including a definition of the ACID properties. Next, it describes D-TX and S-TX our two approaches for supporting transactions in distributed content-based publish/subscribe systems. Finally, it presents the results obtained from an experimental evaluation and concludes with a discussion.

Chapter 7 presents joint conclusions covering the mapping of data-centric workflows into the publish/subscribe abstraction, the geo-distributed execution, and the multi-client transaction approach in distributed publish/subscribe systems.

Chapter 2

Related Work

Business process management (BPM) is a broad field of research spanning various areas and including a wide range of methods, techniques, and tools for designing, managing, analyzing, and executing business processes. An overview about state-of-the-art BPM approaches is provided in [95].

In particular, the various aspects on business process modeling have been subject to extensive research. An overview of business process modeling approaches is presented in [65]. On a first level, the authors distinguish existing languages along two dimensions: first, the purpose the language serves, i.e., description, analysis, or enactment of a process, and second, the view on the process the language provides, i.e., functional, behavioral, informational, or organizational view. Orthogonal to these feature considerations, the authors classify modeling languages into four broad but distinct categories following the scientific and professional traditions established over the past years:

1. *Traditional process modeling languages* focus on understandability of the process by people, i.e., provide different views on the process to give an intuitive description of the relevant aspects. Examples include Petri Nets, Event-driven Process Chains (EPC) [48], or Role Activity diagrams [75].
2. *Object-oriented languages* aim at unifying the descriptions for both business

domain experts and IT experts by *naturally* modeling the world through objects and their relations. However, it turned out that such approaches tend to focus more on the software and IT aspects, either due to inherent shortcomings, or due to explicit focus of the language. Examples in this category include the Unified Modeling Language (UML) and its extensions [71].

3. *Dynamic process modeling languages* contain industry specifications and standards to give a dynamic view on the process, provide a serialization format for exchange, and focus on the full spectrum of usage. For example, process description for human consumption, e.g., BPMN [72], process and workflow exchange, e.g., XPDL [102], or process enactment, e.g., BPEL [70].

4. *Process integration languages* include languages to represent the integration of processes from two or more business partners. The focus lies on describing technology-independent mechanism, like interfaces and exchange formats, to enable integration of different workflow management systems. An example is the Web Services Choreography Description Language (WS-CDL) [103].

Often, organizations maintain multiple, slightly different variants of a process, e.g., sales processes for individual products, which traditional modeling approaches do not explicitly support. Therefore, a plethora of work has also been done in business process variability modeling, which is surveyed in [84]. Commonly, such approaches are characterized by extending conventional models with constructs to capture customizable processes. A variant of the customizable process model for a specific scenario can then be derived by adding or deleting fragments according to a given configuration.

Flexibility in business processes is one of the most active areas of research BPM and has a strong relationship to business process variability modeling [83]. In general, there are three phases in the lifecycle of a of a customizable process model [84]. First, at *design time*, all variants of the model are captured in the customizable process model. Next, at *customization time*, a single variant is configured and extracted from the customizable model representing a subset of processes that can be realized accordingly. Finally, at *run-time*, the customized process model is executed for

individual process instances. While variability primarily concerns the design-time and customization-time phases, flexibility focuses on the run-time phase.

A survey of contemporary approaches in flexibility modeling including a taxonomy and classification is presented in [87]. The authors further distinguish flexibility into four types:

1. *Flexibility by design* refers to specifying multiple alternate execution paths in the process model at design time, which allows the selection of the most suitable path at process execution time (e.g., by choice, parallelism, or interleaving of activities).
2. *Flexibility by deviation* refers to runtime deviations from the original process specification without altering the process model. Deviations encompass changes to the execution within a particular process instance by altering the activities that are to be executed next, i.e., deviating from the original control-flow. The process model itself remains unchanged. In activity-centric languages, this can be supported by specific deviation operations (e.g., undo, redo, skip of activities), or by constraint violations in declarative languages (e.g., not adhering to a previously specified constraint: *A preceeds B*).
3. *Flexibility by underspecification* refers to the ability of executing incomplete process specifications, i.e., process definitions without sufficient information to enable complete execution (e.g., due to placeholders for process fragments). This type of flexibility does not require the model to change at runtime but only to complete the missing parts. *Late binding* refers to the selection of the missing process fragment from a set of available fragments, and *late modeling* requires the construction of a new process fragment at runtime (either from scratch or from existing process fragments).
4. *Flexibility by change* refers to the modification of a process model at runtime such that either a single or all of the currently executing process instances are migrated to a new process model. In contrast to the aforementioned types, here, the design-time process model is changed. Concrete approaches need to specify whether changes affect only new or also concurrent instances, the moment when changes are allowed, and a migration strategy for running process instances.

ADEPT2 [81] is a framework for enabling adaptive process aware information systems by supporting a set of dynamic change patterns. It is based on an activity-centric modeling approach but achieves a higher degree of expressiveness due to several modeling extensions and relaxations [82].

Research has shown that activity-centric modeling languages only provide limited capabilities for realizing the various types of flexibility considering, for instance, the requirements of knowledge-intense work and case management [62]. One reason is that activity-centric approaches only consider data as input or output of activities. Emerging data-centric approaches such as artifact-centric models [69, 23] give data a more central role by introducing lifecycles and stateful date. A survey on handling data in business process models together with a study on the need for more data-awareness is provided in [63]. The authors of [51] summarized artifact-centric modeling approaches.

2.1 Data-centric business artifacts

The existing artifact-centric modeling approaches can be set into the context of a four-dimensional framework, referred to as the BALSA framework, which represents *business-artifacts*, *lifecycles*, *services*, and *associations* [51]. By varying the model and constructs used in each of the four dimensions, different artifact-centric business process models with different characteristics can be obtained [36].

An *artifact* represents a concept to record all information that are necessary to perform the relevant business operations in order to achieve some business objective. It clusters process and data, contains an *identity* to distinguish it from other artifacts, and a *content*, represented as a set of attributes, to describe the artifact state. Attributes are created, deleted, or updated as part of the process activities. An *information model* is used to describe the content and a *lifecycle model* describes all possible, key business-relevant stages, which the artifact might undergo from initialization to completion in reaction to events and services that interact with the artifact. The lifecycle can be specified using flow-charts, finite state machines, state

charts, or declarative mechanisms. *Services* describe business tasks or activities performed by the artifact, which result in an update of the artifact and/or a state change in the information model. *Associations* express the relations among services and artifacts together with their constraints. Constraints specify conditions on when a service is executed. They can be either described procedurally or declaratively. [51]

Our work is also based on the data-centric *business artifacts* paradigm [69, 14, 20], with the Guard-Stage-Milestone meta-model [23, 38, 37] being a natural evolution from the earlier practical artifact-centric meta-models [21, 91], but using a declarative basis and supporting modularity and parallelism within artifact instances. The existing work on GSM operational semantics already addresses some sort of parallelism [92] but does not consider the distributed execution of business artifacts [23]. Notably, GSM also significantly influenced Case Management and the CMMN standard [62, 73].

Recently, other data-centric approaches relying on the artifact-centric paradigm have been proposed. These include the *FlexConnect* meta-model [79], in which processes are organized as interacting business objects, or the AXML Artifact Model [2, 3], which is based on a declarative form of artifacts using Active XML as a basis [1]. ACP-i [104] is another artifact-centric business process model representing an extension to the ACP model [68] to support inter organizational business processes. ACP-i distinguishes artifacts into local artifacts belonging to an organization and shared artifacts, which are commonly agreed for coordination. ArtiNets, to the contrary, is closely related to Petri nets. Here, artifacts form the tokens and the transition firing rule is based on regular expressions and counting constraints [50].

Another object-aware framework that aims at unifying process and data is *PHILharmonicFlows* [52]. Here, workflows are modeled as micro processes that represent the data and behavior of individual objects and macro processes that represent the interactions among such objects.

2.2 Distributed workflow execution

There exists a body of work focused on various aspects of distributed workflow execution. For instance, [12] has a similar goal as our approach for fragmentation and distribution but is applied to an inherently activity-centric workflow model, in which data is only considered as input and output of flow activities (dataflow) and no data-centric execution is supported. This is also true in [11], in which scheduling of workflows in self-organizing wireless networks is addressed to respect resource allocation constraints and dynamic topology changes, or for [88, 55] that use pub/sub techniques to implement some of the BPM execution aspects.

Distributed workflow execution has been studied in the 1990s to also address scalability, fault resilience, and enterprise-wide workflow management [6, 101, 66]. However, these works mostly rely on procedural models and do neither focus on data-centric processes nor support declarative models.

A detailed design of a distributed workflow management system was proposed in [6]. The work bares similarity with our approach in that a business process is fully distributed among a set of nodes. However, the distribution architectures differ fundamentally. In our approach, a content-based message routing substrate naturally enables decoupling, dynamic reconfiguration, system monitoring, and run-time control. This is not addressed in the earlier work.

A behavior-preserving transformation of a centralized activity chart, representing a workflow, into an equivalent partitioned one is described in [66] and realized in the MENTOR system [101]. MENTOR is inspired by compiler-based techniques, including control flow and data flow analysis, in order to parallelize the business process [67]. However, these approaches are complementary to our work since we operate with the original workflow model without analyzing the process. An advantage of executing an unmodified process is that dynamic changes to the executing process instances are possible, as their structure remains unchanged from the original specification.

A distributed workflow execution architecture for BPEL processes is presented

in [28, 55]. This work bares similarity with our approach as it also relies on a pub/sub middleware to coordinate the control-flow. However, the authors do not consider process flexibility and data-locality. Another distribution approach for BPEL based on stream processing units (SPUs) is presented in [10]. SPUs are abstractions for business functions and encapsulate their logic in the context of a business process. SPUs are similar to WFUs in our approach in a sense that they form the unit of distribution. However, they do not support data-centric workflows and build on stream processing primitives instead of pub/sub.

The Global Data Synchronization Network (GDSN) [30] is an industry standard and system implementation that allows companies to integrate data provided by other organizations into their processes. GDSN is used, for instance, by Walmart [99], to access product data of its suppliers and update its own systems. GDSN also relies on a pub/sub middleware to exchange and update data. However, its sole focus is data exchange and not the execution of workflows.

An approach to integrate existing business processes as part of a larger workflow is presented in [18]. The authors define event points in business processes where events can be received or sent. Events are filtered, correlated, and dispatched using a centralized pub/sub model. The interaction of existing business processes is synchronized by event communication. This is similar to our work in terms of allowing business processes to publish and subscribe. In our approach, activities in a business process are decoupled, and the communication between them is performed in a content-based pub/sub broker network.

Hens *et al.* present a distributed approach for procedural, cross-organizational business processes [32]. A BPMN model is split into multiple control-flow fragments, where each fragment is a grouping of activities that belongs to an organizational unit. Each fragment is executed by a dedicated process engine assumed to be hosted by the IT infrastructure of the corresponding organization. Similar to our approach, the communication is based on publish/subscribe. The individual engines subscribe to events that trigger the process fragments they are executing and generate notifications to indicate completion of a fragment to other engines. In contrast to our work, the approach exploits procedural instead of data-centric models; furthermore, the focus

is on the control-flow, and data is only assumed to be transferred with the sequence flow.

The Amit situation manager [4] specifies a language and a centralized execution mechanism in order to reduce complexity in active databases, which are closely related to rule-based execution of data-centric workflows. The situation concept extends composite events in its expressive power, flexibility, and usability; thus, baring some similarity with our notion of WFUs, which are partly realized by subscribing to update events. Amit, however, is a centralized system and focuses on rather isolated situations less complex workflows or interactions among situations.

There is also some work on service orchestrations for business process automation. For example, the ACSI service hub [58, 15] executes workflows specified in GSM and is tailored towards web-service orchestrations. ACSI is built on top of Barcelona [31] as an underlying engine. Roles allow to restrict attribute access to certain institutions at design and runtime. However, Barcelona is a centralized system and maintains all data at a single machine. Similarly, ACP [68] is a centralized engine to automate artifact-centric processes over service-oriented architectures; only the application logic (i.e., the services) are distributed.

A decentralized service orchestration architecture based on the Object Modeling System (OMS) is proposed in [43]. Data that is generated by a particular service is directly forwarded to the subsequent service without being passed through a centralized orchestration engine. Though focusing on the decentralization of data flow to enhance performance, the approach is not data-centric in a sense that the control flow is defined over data, which reduces its flexibility.

Another system that focuses explicitly on scientific workflows is proposed in [74]. This approach is inspired by the work on relational algebra and enables automatic optimization by leveraging a parallel execution model. Workflows are represented as compositions of algebraic operations. The focus of the approach is on handling large collections of data (baring some similarity with MapReduce-style processing).

2.3 Transactions in message-oriented middleware

Relevant related works in the context of transactions in message oriented middleware can be classified into three categories:

1. Transactional message queues realized using centralized brokers [9, 77, 19, 40].
2. Middleware-mediated transactions [57, 56, 94].
3. Transactional messaging in distributed broker architectures [33, 35, 64, 89, 97].

Transactional message queues include proprietary systems like TIBCO's Enterprise Message Service [19], ActiveMQ [9] based on the Java Message Service (JMS), or RabbitMQ [77] based on the Advanced Message Queuing Protocol (AMQP). All systems rely on point-to-point communication or on a single message broker to deliver messages to subscribers. Some approaches like RabbitMQ also enable distribution by clustering message channels. A transaction defines a context, which is used to group a set of messages that need to be atomically sent and received. This set of operation is issued by a single publisher and buffered. On commit of the transaction, messages are delivered to subscribers; otherwise, a rollback is performed and messages are discarded. For instance, TIBCO [19] employs a subject-based pub/sub model and uses 2-phase commit (2-PC) to atomically publish or consume messages on a set of subjects. There are also some database systems providing a pub/sub interface and similar transactional mechanisms. In Redis [80], for instance, a transaction groups a set of operations and executes it atomically and isolated.

Although, the systems bare similarity with our work, i.e., atomic delivery, they significantly differ as they neither support distributed brokers, nor do transactions encompass a mixture of publications and subscriptions by different clients.

Middleware-mediated transactions integrate message queues and distributed object transactions [57].

X^2TS [56] is based on topic-based pub/sub and integrates CORBA's Object Transaction Service and Notification Service to provide transactional guarantees for

multicasting. Similar to message queues, an implicit transaction context is propagated with messages and 2-PC is used for atomic commitment but without compensation. X^2TS contains a caching mechanism at the broker to provide different levels of visibility, which enables a recipient to process a message before the transaction is committed and only get notified about an abort afterwards.

D-Spheres focuses on operationally grouping distributed object transactions [94]. It uses point-to-point communication and allows the descriptive specification of producer/consumer dependencies. Atomicity is provided by 2-PC and a compensation mechanism cancels enqueued messages of aborted transactions.

Compared to our work, both above approaches do not support distributed message routing and transactional combinations of publications and subscriptions by different clients.

There is also a bunch of approaches dealing with transactional guarantees in distributed broker architectures:

In the approach by Hill *et* al. [33], a publisher can request a reply as part of its publication from subscribers. Receiving subscribers then decide if, and which type of reply they want to send (e.g., acknowledgment or result). Replies are routed on the reverse paths of publications and are presented to the publisher using a reply view. The work is motivated by combining the decoupling and scaling features of pub/sub with request/response requirements of certain application and is similar to our acknowledgment mechanism in D-TX.

The Hermes Transaction Service (HTS) supports transactions in content-based pub/sub [97]. HTS is built on top of Hermes, a topic-based system using rendezvous-based routing in broker network [76]. A transaction demarcates the process of generating one or more events at a publisher, the set of events, and a set of processes that are executed at subscribers on consuming these events. HTS supports compensatable transactions; a transaction service creates a transaction context, delivers events together with the context, and provides atomicity through 2-PC. If the transaction aborts, HTS ensures that the operations performed by subscribers in reaction to receiving an event are compensated.

A system build on top of HTS is TOPS [89]. An interesting feature added in TOPS is the support for distributed transactions, i.e., a type of transaction that allows multiple clients to publish as part of the same transaction.

Although, both works share similarities with our approach, neither HTS nor TOPS support subscriptions or modifications to the routing state as part of the transaction. Also isolation of transactions is not considered.

An approach to transactional client mobility in content-based pub/sub is presented in [35]. A transaction encompasses the migration of a client from one broker to another to enable dynamic system adaptation. The protocol is based on 3-phase-commit and compensation but fundamentally differs from our approach. The focus lies on transferring the client state and adapting the routing state according to the new edge broker. No general-purpose transaction model is defined and supported.

Chapter 3

Background

In this chapter, we present background information relevant to understand the approaches presented in this work. In the first section, we cover the Guard-Stage-Milestone (GSM) meta-model for declaratively specifying the lifecycle of business artifacts. GSM forms an abstraction of flexible business processes and provides the basis for our fragmentation and distribution approaches presented in Chapters 4 and 5, respectively. In the second section, we briefly cover concepts and techniques related to distributed transaction processing. These approaches originate from the database community and form the basis of our multi-client publish/subscribe transaction approach presented in Chapter 6.

3.1 The Guard-Stage-Milestone meta-model

Business artifacts are key-conceptual entities that combine both process and data perspectives to describe how business processes navigate through a set of business activities [14, 20, 69]. The Guard-Stage-Milestone (GSM) [37, 38, 23] meta-model provides a declarative basis to specify the lifecycle of a business artifact. In this section, we summarize the key modeling constructs and the associated execution semantics of GSM, which have also been published in in [23].

3.1.1 Key constructs in GSM

The key constructs in GSM are the *Information model*, *Guards*, *Stages*, and *Milestones*. In addition to these, we also recite relevant related concepts and put them into context. The following descriptions are adopted from [23].

Artifact instance — An artifact instance represents a single conceptual business-relevant entity of a particular type progressing through a variety of business operations or activities (e.g., a design-to-order application or the treatment of a patient). Instances can be long-lasting and might exist for months or even years.

Information model — The information model represents all business relevant information about an artifact instance including *data attributes* and *status attributes*. The data attributes represent application-level data about the actual business, e.g., customer or patient information, legal documents, etc. The status attributes represent information about the internal state of the artifact, i.e., its progress in the lifecycle, which is indicated by the current status of milestones and stages.

Task — A task refers to a business-relevant piece of work, i.e., an activity, that is supposed to be performed by an outside agent (e.g., a web-service, a human worker, or an actuator device). On the one side, the artifact instance invokes tasks and provides input data from its information model. On the other side, it incorporates data updates into the information model whenever the artifact instance notices the termination of a task.

Environment — The environment represents the ecosystem, in which the artifact instance exists and hosts the task executions of the artifact.

Event — There are three different categories of events is GSM:

1. *Outgoing events* represent event messages that are sent from the artifact to its environment. For instance, a *task-invocation event* is supposed to invoke a particular task, possibly, together with current values of data attributes from the information model (also referred to as 2-way service call).

2. *Incoming events* represent event messages that are sent from the environment to the artifact. *Request events*, for instance, are sent pro-actively, e.g., a user request to create a new artifact instance (also referred to as 1-way service call). *Task-termination events*, in contrast, are sent whenever a previously invoked task has completed its execution and, possibly, contains updates to data attributes from the information model.

3. *Internal events* or *status events* refer to status changes within the artifact, such as a milestone or stage changing its state from false to true, or vice versa.

Sentry — A sentry is an expression in a condition language that can refer to incoming events, internal events, and data or status attributes. Sentries provide the core of the ECA-like rules that drive the artifact evolution in GSM. A sentry has the form **on** < *event* > **if** < *condition* > **then** < *action* >, where either the event or condition may be omitted.

Milestone — A milestone is a business-relevant objective within the artifact, which is represented as a named Boolean attribute. A milestone can be either *achieved*, i.e., true, or *invalidated*, i.e., false. The milestone status is also reflected as a status attribute in the information model. An achieving sentry determines when the milestone becomes achieved and turns to true. In addition, a milestone can also have an invalidating sentry, which determines when the milestone turns to false and becomes invalidated.

Stage — A stage refers to a cluster of business-relevant activity supposed to be executed during artifact evolution. Stages may form hierarchies for providing a rich form of modularity, but only atomic stages contain tasks—in particular, exactly a single task. Every stage is represented as a named Boolean attribute in the information model. It can be either *opened* or *active*, i.e. true, which corresponds to when the stage is currently executing, or *closed* or *inactive*, i.e., false, which corresponds to when the stage is currently not executing. In addition, every stage *owns* one or more milestones and, intuitively, the purpose of a stage is to achieve one of its milestones. A single stage can only have a single executing occurrence at a time but different stages can execute concurrently, which enables parallel task executions.

Guard — A guard is associated with a stage and controls when to open the stage and start its execution. It is specified as a sentry and remains unnamed in contrast to milestones and stages.

Lifecycle model — The lifecycle model is a component representing all stages and milestones of an artifact. It contains the hierarchy among stages, the associations of stages with guards and milestones, the relationship of tasks to atomic stages, and the specifications of sentries.

3.1.2 GSM operational semantics

To understand the operational semantics of GSM, also referred to as execution semantics, we briefly cover a set of relevant related concepts first. The following descriptions are adopted from [23].

The GSM operational semantics assumes that only a single incoming event is processed at a time. If more events occur simultaneously a global event queue in the GSM system is assumed to manage incoming events. Furthermore, during execution, the GSM meta-model maintains two invariants corresponding to an intuitive understanding of the interactions among stages and milestones:

GSM-1 For a stage S that owns some milestone m, it is not allowed that both S is active and m is achieved. More precisely, if S becomes active m must change its status to false, and if m changes its status to true then S must become inactive.

GSM-2 For a stage S that becomes inactive, all of its substages *S'* also become inactive.

During execution, at a given time, a complete description of an artifact instance, including the values of all data and status attributes from its information model, is captured as a *snapshot* of the GSM system, also denoted Σ. The execution of an artifact instance during its lifetime can be considered as a sequence of such snapshots,

	Basis	Prerequisite	Antecedent	Consequent
PAC-1	Guard: for each stage S, for each guard φ of S. (Include term $active_{S'}$ if S' is parent of S.)	$\neg S$	$\varphi \wedge S'$	$\oplus S$
PAC-2	Milestone achiever: For each milestone m of stage S with achieving sentry φ.	S	φ	$\oplus m$
PAC-3	Milestone invalidator: For each milestone m of stage S with invalidating sentry φ.	m	φ	$\ominus m$
PAC-4	Guard invalidating milestone: For each guard φ of a stage S, for each milestone m not occuring in a toplevel conjunct $\neg m$. (Include term S' if S' is parent of S.)	m	$\varphi \wedge S'$	$\ominus m$
PAC-5	For each milestone m of a stage S.	S	$\oplus m$	$\ominus S$
PAC-6	For each stage S child of S'.	S	$\ominus S'$	$\ominus S$

Table 3.1.1: Templates for PAC rules associated with a GSM model (adopted from [23]).

each produced on receiving an incoming external event from its environment and incorporating its impact.

Internally, the reaction of the GSM system on some incoming external event is summarized as a *Business Step*, or B-step for short, which corresponds to the smallest, atomic, business-relevant change within the artifact. Intuitively, a B-step captures the firing of all sentries applicable to the current snapshot after receiving an external event.

With respect to the actual execution of a GSM model, we focus on the incremental formulation of the GSM operational semantics: a variation of incremental firing of Event-Condition-Action (ECA) rules, known as Prerequisite-Antecedent-Consequent (PAC) rules in GSM. Essentially, PAC rules capture the behavior of sentries and differ from traditional ECA rules in a way that they also incorporate a temporal component, i.e., the prerequisite, which allows for the specification of conditions on a prior system snapshot. The GSM meta-model differentiates six distinct types of PAC rules that are also described in Table 3.1.1: PAC-1 for achieving guards; PAC-2 for achieving milestones; PAC-3 for invalidating a milestone once its stage is opened; PAC-4 for invalidating a milestone once its invalidating sentry is achieved; PAC-5 for closing a stage when one of its milestones is achieved; and PAC-6 for closing a substage when its parent stage is closed. The set of PAC rules can be derived in polynomial time from a GSM model.

The order of PAC rule firings within a B-step is defined by the generalized notion of the Polarized Dependency Graph (PDG). The PDG imposes a topological sort order on PAC rule firing, essentially a PAC rule stratification, in which no cyclic relation among PAC rules is allowed, which requires the PDG graph to be acyclic. The PDG imposed order on rule firing guarantees the uniqueness and the termination properties in the context of defining the smallest logical unit of work, i.e., a B-step, as the well-formedness of a finite set of PAC rules within the B-step.

The polarized dependency graph for a GSM model $\Gamma = (\mathcal{I}, \mathcal{R})$, denoted $PDG(\Gamma)$, where $\mathcal{R}$ is the set of PAC-rules and $\mathcal{I}$ is the information model of Γ, is constructed as follows: For each status attribute s in $\mathcal{I}$, there are two nodes $\langle +, s\rangle$ and $\langle -, s\rangle$. For each stage S and each of its guards φ, there is a node $\langle +, S.\varphi\rangle$. For the description of the edges of $PDG(\Gamma)$, the antecedent α of a PAC rule is written as $\tau \wedge \gamma$, where τ is either empty, or has the form `latestIncEvent` $=$ `E` for some external event type `E`, or has the form $\odot s$, i.e., status event, for some status attribute s, where $\odot \in \{\oplus, \ominus\}$, referring to a positive or negative toggling of s, and γ contains no external event types or status events.

1. For each PAC-1 rule $\langle \neg s, \tau \wedge \gamma, \oplus s\rangle \in \mathcal{R}$:
   - If $\hat{\odot} a$ is a toggling status attribute occurring in τ, include a directed edge $(\hat{\odot} a, +S.\varphi)$.
   - If status attribute a occurs in γ, include two directed edges $(+a, +S.\varphi)$ and $(-a, +S.\varphi)$.
2. For each guard φ of stage S represented by status attribute s:
   - Add edge $(+S.\varphi, +s)$.
   - For each milestone m owned by S that does not occur in a top-level conjunct of form $\neg m$ in γ, add the edge $(+S.\varphi, -m)$.
3. For each PAC rule $\langle \pi, \tau \wedge \gamma, \odot s\rangle$ from templates PAC-2 or PAC-3 $\in \mathcal{R}$:
   - If $\hat{\odot} a$ is a toggling status attribute occurring in τ, include a directed edge $(\hat{\odot} a, \odot s)$.

- If status attribute a occurs in γ, include two edges $(+a, \odot s)$ and $(-a, \odot s)$.

4. For each PAC-5 rule $\langle s, \oplus m, \ominus s\rangle \in \mathcal{R}$, where s represents a stage S, add edge $(+m, -s)$.

5. For each PAC-6 rule $\langle s, \ominus s', \ominus s\rangle \in \mathcal{R}$, where s' represents a stage S' being a child of stage S, add edge $(-s', -s)$.

The incremental formulation of GSM, in turn, the execution of PAC rules in the prescribed order of the PDG, is driven and initiated upon receiving an external event from the environment. The set of all applicable PAC rules is then executed in response to receiving the external event and the firing of these PAC rules is sequenced to form a B-step. The semantics of a B-step with respect to the overall GSM system snapshot is summarized using a 5-tuple (cf. Figure 3.1.1).

$$\langle \Sigma, e, t, \Sigma', Gen\rangle$$

Here, Σ is the current system snapshot of the GSM instance prior to consuming the external event e at logical time t, Σ' is the new snapshot of the system after firing all relevant PAC rules that are triggered directly or indirectly by the external event e, and Gen is a set of generated immutable events as a result of 1-way and 2-way service calls that may be encapsulated in a task.

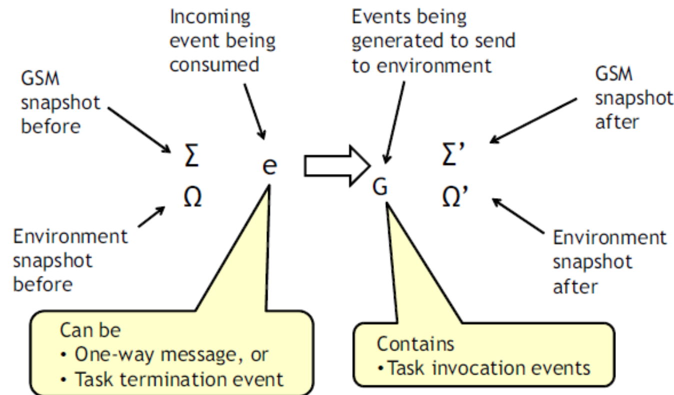


Figure 3.1.1: Illustration of GSM B-step (adopted from [23])

.

3.2 Distributed transactions

Transactions have mostly been studied in the area of database systems. A database transaction is defined as a sequence of atomic operations on data objects forming an atomic unit of work. Operations are typically issued by a single client together with a distinct *begin* and *end* operation and executed on one or multiple severs, while providing atomicity, consistency, isolation, and durability—also referred to as the *ACID* properties [100].

- *atomicity*—a transaction executes completely or not at all, i.e., either all operations are applied or none of them.
- *consistency*—a transaction transfers the system from a correct state into another correct state. A correct system state is defined by a consistency model, which is enforced by scheduling all operation according to a specific order and maintaining database invariants.
- *isolation*—concurrent transactions do not influence each other. An isolation level defines, when changes applied by a transaction become visible to other transactions.
- *durability*—committed operations, as part of a transactions, survive node or network failures.

Within a database management system, the ACID properties are maintained by a transaction manager containing at least: (1) a *recovery* mechanism to guarantee atomicity, (2) a *scheduler* to guarantee consistency, and (3) a *concurrency control* mechanism to guarantee isolation. [100]

3.2.1 Consistency models and consistent replication

As part of the "C" in the ACID properties, which requires a transaction to transition from on correct system state into a new correct system state, consistent replication

is a challenge particular arising in distributed databases. Here, a consistency model defines a contract between a distributed data store and a set of processes to specify the result of read and write operations in the presence of concurrency. [100] According to these guarantees, consistency models can be classified and compared. A consistency model C_S is stronger than a consistency model C_W, if C_S provides all guarantees of C_W and more.

Strict consistency is the strongest model and requires that any write by any process is instantaneously visible to all other processes. In practice, strict consistency cannot be achieved efficiently. A slightly weaker model is *sequential consistency*, which requires that writes to variables by different processes have to be seen in the same sequential, (total-) order by all processes. *Causal consistency* is relaxing this requirement to causally related operations that must be seen in the same order.

Recently, a new consistency model called *Causal+ consistency*, residing between sequential and causal consistency, has been identified to describe the replication behavior in key-value stores [59]. Causal+ consistency is defined as causal consistency with convergent conflict handling. Causal consistency builds upon the concept of *potential causality*, which is denoted with $\leadsto$ and defined by three rules:

1. **Thread**—if a and b are two operations on a single thread of execution, then $a \leadsto b$ if a happened before b.
2. **Reads from**—if a is a write operation and b is the operation reading the value written by a, then $a \leadsto b$.
3. **Transitivity**—for operations a, b, and c, if $a \leadsto b$ and $b \leadsto c$, then $a \leadsto c$.

Causal consistency does not order concurrent operations: two unrelated operations a and b can be executed (replicated) in any order. However, if two unrelated operations a and b write to the same variable, they are in conflict. Conflicts might lead to diverging replicas or unexpected system behavior. Therefore, in causal+ consistency, *convergent conflict handling* defines that conflicting writes are handled in the same way by different replicas. This is achieved by leveraging a commutative

and associative handler function (h), which ensures that applying conflicting writes in different orders eventually results in the same state. Typically, a *conflict-free replicated data type* (CRDT) is used to implement this behavior. A CRDT can be mutated instantaneous and concurrently and any potential divergence is guaranteed to be eventually eliminated [5].

Eventual consistency is the weakest consistency model and only guarantees that all replicas eventually reach the same state if clients stop submitting updates. As concurrent updates propagate asynchronously through the system this might result in conflicts due to violated constraints. CRDTs have also been applied to implement eventual consistency without requiring roll-backs. Diverging states at replicas are avoided by leveraging suitable handler functions h, for instance, *add()* or *remove()*, to ensure that the application of conflicting writes in different orders eventually results in the same state.

In our D-TX approach for supporting transactions in distributed publish/subscribe systems, we provide sequential consistency by imposing a total order on pub/sub operations. In our S-TX approach, to the contrary, we leverage the CRDT nature of routing tables in distributed publish/subscribe systems to achieve sequential consistency.

3.2.2 Isolation levels

Isolation levels are classified according to their capability in avoiding unwanted concurrency effects that occur when reading data objects, e.g., *dirty reads*, *phantom reads*, or *non-repeatable reads.* In general, lower isolation levels increase the chances for these concurrency effects [100].

Serializability is the highest isolation level and avoids concurrency effects completely. The result of concurrently executing two transactions is the same as if both transactions are executed one after the other. Serializability is either achieved by read/write locks on objects, potentially delaying other transactions (pessimistic), e.g., 2-phase locking (2PL), or by detecting conflicts at commit time and potentially aborting

transactions very late (optimistic), e.g., snapshot isolation.

Repeatable Reads is a little weaker than serializability and allows phantom reads. Repeated read operations on selected objects always return the same result because read and write locks are maintained for the complete duration of the transaction. However, since range-locks are not maintained phantom-reads are possible for repeated range queries.

Read committed, in contrast to repeatable reads, only maintains write locks on selected objects for the complete duration of the transaction. Read locks are released once the read operation has been executed, which might also lead to the non-repeatable reads phenomenon in addition to phantom reads.

Read uncommitted, to the contrary, is a very low isolation level that reveals all three read phenomena. It allows a transaction to read data objects that are written but not yet committed by another transaction.

3.2.3 Distributed atomic commitment

A distributed transaction involves the access to transactional resources on multiple hosts in a networked environment, e.g., a database that is partitioned or replicated across several servers. Here, the transaction manager synchronizes the enforcement of the ACID properties among all participating database servers.

The core challenges in distributed transactions originate from the atomicity and isolation properties. Both require distributed agreement: atomicity, in order to decide whether to commit or abort the transaction; and isolation, to either agree on the conflict/commit order, or to propagate locks on data objects. A well-established solution to solve this problem is the *2-phase commit* protocol (2-PC) [100] combined with *commitment ordering* [78] to impose an order on transactions. Our D-TX approach adopts these concepts to distributed publish/subscribe systems for managing multi-client transactions.

2-phase commit (2-PC) is a distributed atomic commitment protocol, which is initiated by a dedicated transaction coordinator (TXC) among a set of database servers to find agreement [13]: In a *voting phase*, the TXC sends a `commit-prepare` message to all servers. Each server executes the operations up to a point where it can commit the transaction by adding entries to both undo and redo logs. Then it replies with either `vote-commit`, if all operations succeeded, or `vote-abort`, if an error or conflict occurred. Once the TXC received commit votes from all servers, it starts with the *commit phase* by sending a `commit` message that causes all servers to complete their operations, release resources, and acknowledge the completion. On receiving acknowledgments from all servers the TCX completes the transaction itself. However, if the TXC received at least a single abort vote in the voting phase, it sends a `rollback` message that causes all servers to undo the operations using their undo log and acknowledge the rollback. Again, once the TXC received all acknowledgments it completes the transaction.

Commitment ordering (CO) refers to a property of histories (i.e., schedules of committed transaction) that guarantees serializability, and describes a class of techniques to achieve global serializability across a set of distributed data bases [78]. It allows the implementation of an optimistic concurrency control mechanism by generating a commitment schedule that is compatible with a precedence order imposed on (concurrent) transactions. More formally, a CO scheduler guarantees that for any two conflicting transactions T_1, T_2, if T_1 precedes T_2, than T_1 commits before T_2.

Chapter 4

Publish/Subscribe Mapping of Data-centric Workflows

In this chapter, we present our formal fragmentation approach for data-centric workflows, represented in GSM, to establish the foundation for workflow distribution according to geographical and/or legal constraints and enable their parallel execution. The approach is based on mapping key GSM constructs like PAC rules, the information model, and the B-step semantics into a set of publications and subscriptions. We start by providing a formal model of data-centric workflows specified in GSM in Section 4.1. Then we present a suitable formalization of the publish/subscribe abstraction by reformulating publications and subscriptions together with their matching, consumption, and notification policies in Section 4.2. In Section 4.3, we provide an overview of our mapping functions and in Section 4.4 we particularly focus on the definition of the necessary set of subscriptions. Next, in Section 4.5, we proof the equivalence of our pub/sub formulation with the original GSM execution semantics and quantify its execution overhead in terms of generated messages. Finally, we analyze the complexity of optimal workflow distribution by formalizing the problem as the colored multiway cut problem in Section 4.6. We show the intractability of the problem by reduction from the multiway cut problem and provide a greedy algorithm with a constant factor approximation.

4.1 Formal data-centric workflow model

We begin by describing a formal representation of the GSM meta-model. Then we provide a concrete example of a workflow represented in GSM, which will serve as a running example throughout the remainder of this chapter.

4.1.1 Overview of GSM meta-model

A GSM model describing a workflow is defined as a set of artifact types with lifecycle, denoted by $\mathcal{A}$, where each artifact $\mathcal{A}$ is defined by a six-tuple:

$$\mathcal{A} = \langle x,\ Att,\ Typ,\ Stg,\ Mst,\ Lcyc \rangle. \tag{4.1.1}$$

As indicated in Chapter 3, a GSM model can succinctly be described as the grouping of business processes into an artifact type $\mathcal{A}$ that corresponds to an actual business entity within an organization. Each artifact is comprised of a set of goal-oriented work items with lifecycles, in which a work item is modeled as stages (Stg) and goals are referred to as milestones (Mst). In addition, each artifact may have many instances (x), i.e., workflow instances or enactments over a globally shared information model in order to store relevant data, i.e., a set of data and status attributes (Att) and their associated data types (Typ). Moreover, the lifecycle model, i.e., the blueprint for the artifact's evolution through its various stages, is given by:

$$Lcyc = \langle Substage,\ Tasks,\ Owns,\ Guards,\ Ach,\ Inv \rangle. \tag{4.1.2}$$

The lifecycle of each stage captures the hierarchy of its substages ($Substage$), encapsulation of a task within each (sub)stage ($Tasks$), information about stage nesting ($Owns$), conditions for enabling (sub)stages ($Guards$), conditions for determining the successful completion of (sub)stages (Ach), and conditions for disabling (sub)stages (Inv). Roughly speaking, the GSM meta-model defines a workflow through the lens

of a stage, guards for entering a stage, and milestones for leaving a stage.

A key primitive GSM construct, in addition to guard, stage, and milestone, is *sentry*, which is the building block of guards and milestones. A sentry is a Boolean formula of type $\chi(x)$ that consists of two parts: the *(triggering) event* $\xi(x)$, which is a Boolean formula to test the type of an incoming external event, and a *condition* $\varphi(x)$, which is a Boolean formula defined over a subset of status attributes. A sentry may have three different forms:

(i) `on` $\xi(x)$ `if` $\varphi(x)$

(ii) `on` $\xi(x)$

(iii) `if` $\varphi(x)$

The operational semantics of GSM are centered around the incremental firing of a set of Prerequiste-Antecedent-Consequent (PAC) rules, which encode the behavior of sentries. The order of PAC rule firings is dictated by the order prescribed by the Polarized Dependency Graph (PDG). For more details on PAC rules, their generation based on sentries, and the PDG please refer to Chapter 3. The execution of all applicable PAC rules in reaction to receiving an external event from the environment forms an atomic B-step summarized as a 5-tuple $\langle \Sigma, e, t, \Sigma', Gen \rangle$, where Σ is the current GSM system snapshot prior to consuming the external event e at time t, Σ' is the new system snapshot after firing all relevant PAC rules that are triggered by the external event e, and *Gen* is a set of generated *immutable* events as a result of service calls encapsulated in tasks.

Definition 1. An immutable event is a static instantiation of an event schema such that all its attribute values are predefined and not changed at runtime.

Thus, the B-step is formalized with respect to the sequence of PAC rule firings such that $\Sigma = \Sigma_0, \Sigma_1, \Sigma_2, \cdots, \Sigma_n = \Sigma'$ (where $\Sigma_0 \neq \Sigma_1$). Thus, after applying the ith PAC rule, according to the order imposed by the PDG, the state advances from Σ_i to Σ_{i+1}, which is also referred to as a micro-B-step in GSM.

The key properties surrounding B-steps are that a B-step $\langle \Sigma, e, t, \Sigma', Gen \rangle$ always terminates and ends in a unique state Σ', where $\Sigma \neq \Sigma'$. We refer to these as uniqueness properties of B-steps [23]. They are achieved in part by restricting that each *Att* in the GSM model changes at most once as a result of PAC rules firing within the context of a single B-step (i.e., toggle-once property), which implies that a change cannot be undone, and in part by executing all relevant PAC rules whose consequents are reachable in the PDG and in an order imposed by the PDG; namely, visiting every reachable node in the PDG using a strata-based, breadth-first graph traversal. PAC rules are applicable once their prerequisite and antecedent are satisfied; only then the consequent is applied and the current state of a GSM instance changes.

The GSM execution model assumes a global, external event queue, and the current GSM operational semantics is serialized w.r.t. the external event queue. In this work, we also rely on a global event queue to orchestrate concurrent B-step executions such that the event queue behaves as a pseudo-global clock. However, we interleave and pipeline the processing of multiple B-steps over a loosely coupled, distributed pub/sub infrastructure, in which each B-step is associated with a different external event. We achieve this distributed and parallel execution while guaranteeing an identical behavior as if B-steps were processed centrally and in the sequential order of the global event queue.

For enhanced comprehensibility, we focus on the core aspects of GSM workflows (i.e., the data and the lifecycle) to describe our mapping. Therefore, we formalize a GSM workflow model, Γ, as follows:

$$\Gamma = \langle \mathcal{I}, \mathcal{R} \rangle, \tag{4.1.3}$$

where $\mathcal{I}$ is the workflow information model that consists of a set of ordered $\langle$*attr*, *datatype*$\rangle$-pairs and distinguishes between data attributes ($\mathcal{I}_d$), i.e., application data, and status attributes ($\mathcal{I}_s$) describing the state of the workflow within its lifecycle. The number of status attributes is finite and bounded by the model. $\mathcal{R}$ is a set of acyclic PAC rules representing the lifecycle. The operational semantics of Γ follows the general notion of incremental operational semantics [23].

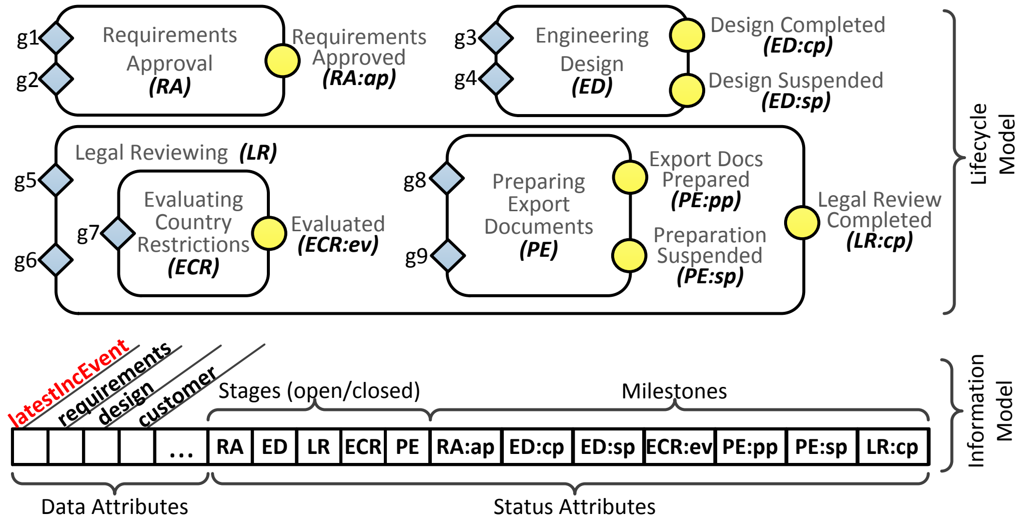


Figure 4.1.1: "Design-to-order" business process modeled in GSM (adopted from [23]).

4.1.2 Example of data-centric workflow in GSM

The example depicted in Figure 4.1.1 represents a data-centric GSM model for a product design process on behalf of an external customer. It is structured into various stages (i.e., rounded rectangles) describing the individual task definitions. Guards are denoted by diamonds and milestones by circles.

Upon a customer order, i.e., an external request event of type `R:NewOrder`, a new workflow is instantiated and the corresponding product requirements are approved. Once the requirements have been approved (i.e., an external task termination event of type `T:RequirementsApproval`, which indicates that the clerk in charge of finished the task, has been received), the actual engineering stage is opened. In case the customer decides to change the requirements afterwards (i.e., `R:CustomerChange`), the design stage is suspended and the requirements are approved again. The legal reviewing of the order is encapsulated in a separate stage comprising two sub-stages. While country restrictions can be evaluated in parallel with the approval of requirements, the preparation of the export documents requires a completed design. Furthermore, preparation is suspended and country restrictions are re-evaluated if requirements change. The whole process is accomplished once the export documents are prepared.

The above behavior is implicitly described by the sentries associated with guards (cf. Table 4.1.1) and milestones (cf. Table 4.1.2) of the GSM model. The triggering events in the sentry definitions can be either external or internal events. Conceptually, external events are further divided into request-events invoking a task (indicated with a "`R`" in the example) and task-termination-events starting with a "`T`". Internal events represent status attribute updates in the information model, whereby $\oplus$ indicates that an attribute toggled to true and $\ominus$ indicates that it toggled to false.

Guard	Sentry
g1	latestIncEvent = "R:NewOrder"
g2	latestIncEvent = "R:CustomerChange"
g3	RA:ap $\wedge$ ECR:ev $\wedge$ $\neg$ ED:cp
g4	latestIncEvent = "R:ResumeDesign"
g5	latestIncEvent = "R:NewOrder"
g6	latestIncEvent = "R:RedoExportDocs"
g7	$\neg$ ECR:ev
g8	$\oplus$ ED:cp
g9	latestIncEvent = "R:RedoExportDocs"

Table 4.1.1: Sentries associated with guards.

Milestone	Type	Sentry
RA:ap	Ach	latestIncEvent = "T:RequirementsApproval"
ED:cp	Ach	latestIncEvent = "T:EngineeringDesign"
	Inv	$\ominus$ RA:ap
ED:sp	Ach	$\ominus$ RA:ap
ECR:ev	Ach	latestIncEvent = "T:EvalCountryRestrictions"
PE:pp	Ach	latestIncEvent = "T:PreparingExportDocs"
	Inv	$\ominus$ ED:cp
PE:sp	Ach	$\ominus$ ED:cp
LR:cp	Ach	$\oplus$ PE:pp
	Inv	$\ominus$ PE:pp

Table 4.1.2: Sentries associated with milestones.

For example, guard `g1` is achieved if the latest incoming external event was of type `R:NewOrder`, which corresponds to a new customer request. Similar, milestone `ExportDocsPrepared` is invalidated if an internal status-update event notified the invalidation of milestone `DesignCompleted` ($\ominus$`ED : cp`, for short).

The PAC rules for this workflow are derived from the GSM model according to the six rule templates described in Table 3.1.1. Altogether, this comprises a set of 41 rules that are depicted in Table 4.1.3. An excerpt of three PAC rules, which will be relevant for a subsequent running example of the mapping is depicted in Table 4.1.4. The order of PAC rule firing for the GSM operational semantics is described by the PDG depicted in Figure 4.1.2. It has been established according to the construction algorithm described in Section 3.1.2.

No	Prerequisite	Antecedent	Consequent
	PAC-1 rules		
1	¬ RA	latestIncEvent = "R:NewOrder"	⊕ RA
2	¬ RA	latestIncEvent = "R:CustomerChange"	⊕ RA
3	¬ ED	RA:ap ∧ ECR:ev ∧ ¬ ED:cp	⊕ ED
4	¬ ED	latestIncEvent = "R:ResumeEngineeringDesign"	⊕ ED
5	¬ LR	latestIncEvent = "R:NewOrder"	⊕ LR
6	¬ LR	latestIncEvent = "R:RedoExportDocs"	⊕ LR
7	¬ ECR	¬ ECR:ev ∧ LR	⊕ ECR
8	¬ PE	⊕ ED:cp ∧ LR	⊕ PE
9	¬ PE	latestIncEvent = "R:RedoExportDocs" ∧ LR	⊕ PE
	PAC-2 rules		
10	RA	latestIncEvent = "T:RequirementsApproval"	⊕ RA:ap
11	ED	latestIncEvent = "T:EngineeringDesign"	⊕ ED:cp
12	ED	⊖ RA:ap	⊕ ED:sp
13	ECR	latestIncEvent = "T:EvalCountryRestrictions"	⊕ ECR:ev
14	PE	latestIncEvent = "T:PreparingExportDocs"	⊕ PE:pp
15	PE	⊖ ED:cp	⊕ PE:sp
16	LR	⊕ PE:pp	⊕ LR:cp
	PAC-3 rules		
17	ED:cp	⊖ RA:ap	⊖ ED:cp
18	PE:pp	⊖ ED:cp	⊖ PE:pp
19	LR:cp	⊖ PE:pp	⊖ LR:cp
	PAC-4 rules		
20	RA:ap	latestIncEvent = "R:NewOrder"	⊖ RA:ap
21	RA:ap	latestIncEvent = "R:CustomerChange"	⊖ RA:ap
22	ED:cp	RA:ap ∧ ECR:ev ∧ ¬ ED:cp	⊖ ED:cp
23	ED:sp	RA:ap ∧ ECR:ev ∧ ¬ ED:cp	⊖ ED:sp
24	ED:cp	latestIncEvent = "R:ResumeEngineeringDesign"	⊖ ED:cp
25	ED:sp	latestIncEvent = "R:ResumeEngineeringDesign"	⊖ ED:sp
26	LR:cp	latestIncEvent = "R:NewOrder"	⊖ LR:cp
27	LR:cp	latestIncEvent = "R:RedoExportDocs"	⊖ LR:cp
28	ECR:ev	¬ ECR:ev ∧ LR	⊖ ECR:ev
29	PE:pp	⊕ ED:cp ∧ LR	⊖ PE:pp
30	PE:sp	⊕ ED:cp ∧ LR	⊖ PE:sp
31	PE:pp	latestIncEvent = "R:RedoExportDocs" ∧ LR	⊖ PE:pp
32	PE:sp	latestIncEvent = "R:RedoExportDocs" ∧ LR	⊖ PE:sp
	PAC-5 rules		
33	RA	⊕RA:ap	⊖ RA
34	ED	⊕ED:cp	⊖ ED
35	ED	⊕ED:sp	⊖ ED
36	LR	⊕LR:cp	⊖ LR
37	ECR	⊕ECR:ev	⊖ ECR
38	PE	⊕PE:pp	⊖ PE
39	PE	⊕PE:sp	⊖ PE
	PAC-6 rules		
40	ECR	⊖ LR	⊖ ECR
41	PE	⊖ LR	⊖ PE

Table 4.1.3: Complete set of PAC rules for the "Design-to-order" workflow.

No	Prerequisite	Antecedent	Consequent
1	PE:pp	⊖ ED:cp	⊖ PE:pp
2	PE:pp	⊕ ED:cp ∧ LR	⊖ PE:pp
3	PE:pp	latestIncEvent = "R:RedoExportDocs" ∧ LR	⊖ PE:pp

Table 4.1.4: Excerpt of PAC rules for "Design-to-order" workflow.

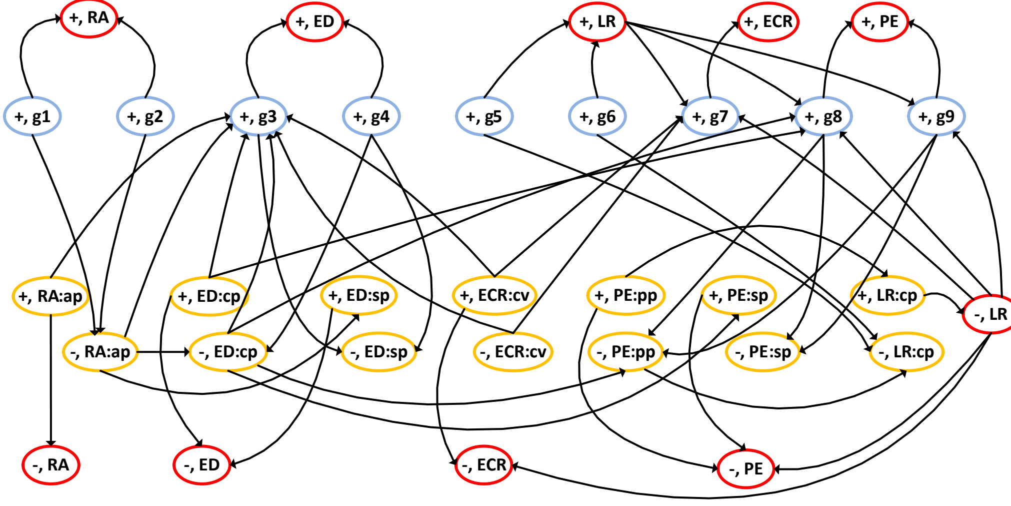


Figure 4.1.2: Polarized dependency graph (PDG) for "Design-to-order" workflow.

In the rest of this chapter, we exploit this example to illustrate our GSM-to-pub/sub mapping. In particular, we show the construction of two distinct subscriptions capturing the semantics of (1) the invalidation of milestone `PE:pp` (based on the rules depicted in Table 4.1.4) and (2) maintaining a consistent view on status attribute `ExportDocsPrepared` (i.e., `PE:pp`).

4.2 Publish/Subscribe schema

In this section, we present the necessary formalization of the pub/sub abstraction for subsequently being able to prove the correctness of our mapping from data-centric workflows to pub/sub. At the core of the pub/sub abstraction lies a set of publications ($\mathbb{P}$) and subscriptions ($\mathbb{S}$). Each publication, $\mathcal{P} \in \mathbb{P}$, is defined as follows:

$$\mathcal{P} = \langle \mathcal{E} \rangle, \tag{4.2.1}$$

where $\mathcal{E}$ defines the publication's event schema that consists of a set of ordered $\langle attr, datatype \rangle$-pairs. Events are instances of this schema and defined as sets of ordered $\langle attr, value \rangle$-pairs, where *value* is an instance of the *datatype* specified in

$\mathcal{E}$. Over time, a publisher continuously produces events that conform to its event schema. Each subscription, $\mathcal{S} \in \mathbb{S}$, is defined as follows:

$$\mathcal{S} = \langle \mathcal{D}, \Phi(\rho_k), \delta(\rho_k), N(\rho_k), \Psi(\rho_k) \rangle \tag{4.2.2}$$

where $\rho_k = \langle e, t, x \rangle$, with event type e, logical event time t, and subscription instance x (x is a context variable essentially identifying a concrete workflow instance).

$\mathcal{D}$ is the data model. It describes the internal state of a subscription and its unique key is formed by the triplet ρ_k.

$$\mathcal{D} = \langle \mathtt{e}, \mathtt{t}, \mathtt{x}, \mathtt{onE_1}, \cdots, \mathtt{onE_m}, \mathtt{d_1}, \cdots, \mathtt{d_n}, \mathtt{s_1}, \cdots, \mathtt{s_p}, \mathtt{visited} \rangle \tag{4.2.3}$$

For every toggling status attribute appearing in antecedents of PAC rules there is a column $\mathtt{onE_i}$ in $\mathcal{D}$. Moreover, there is a column for every data attribute $\mathtt{d_i}$ (i.e., application data) and every status attribute $\mathtt{s_j}$ (i.e., internal workflow state) appearing in logical expressions of PAC rules. The tuple with key ρ_k maintains these values as the result of receiving events associated with ρ_k. The final column `visited` indicates whether or not all the values in this tuple have *stabilized*, i.e., do no longer change as a result of external event ρ_k. Setting `visited` to true in the tuple with key ρ_k implies that this tuple is now a read-only tuple and any notification (event generation) associated with $\mathcal{S}$ for event ρ_k has been completed. A read-only tuple is retained for maintaining an execution history and for enabling parallel and distributed processing of PAC rules. We define the domain range for status attributes as $\mathrm{DOM_{status}} = \{true, false, \varnothing\}$, i.e., the Boolean constants together with a special symbol $\varnothing$. The domain for status changes is defined as $\mathrm{DOM_{toggling}} = \{\langle \mathrm{Boolean} \times \mathrm{Boolean} \rangle, \varnothing\}$, i.e., all possible transitions for the status attribute together with $\varnothing$. Similarly, the domain range for data attributes is defined as $\mathrm{DOM_{data}} = \{\mathrm{String} \cup \mathrm{Number} \cup \varnothing\}$, i.e., any string or number together with $\varnothing$. The special symbol $\varnothing$ indicates that the current value is unstable, i.e., the attribute has not been updated in the context of the external event ρ_k, while all other domain values are considered as stable.

$\Phi(\rho_k)$ is the subscription's matching condition. It is a disjunction over $\phi_i(\rho_k) \in \Phi(\rho_k)$, where each $\phi_i(\rho_k)$ is a condition, that is, a logical formula representing the antecedent of a PAC rule, that is instantiated and correlated with each external event ρ_k. This condition is expressed over the condition language $\mathcal{L}$ that is a subset of First-Order Logic (FOL) supporting: scalar values, binary relations (i.e., logical operators $(\vee, \wedge, \rightarrow)$, relational operators (i.e., $<, \leq, =, \neq, \geq, >$, the unary relation $\neg$, and quantification over subscription instances ρ_k, i.e., $\forall$ and $\exists$). The quantification domain for ρ is totally ordered by time t and instance x. Furthermore, we define the following functions.

1. $\tau_k(attr, \rho_k)$, or simply, $\tau(attr)$, which returns the current value of the attribute $attr$ w.r.t. ρ_k in $\mathcal{D}$.
2. $\tau_{k-1}(attr, \rho_k)$ which returns the last value of the attribute $attr$ w.r.t. ρ_k for $k > 2$ in $\mathcal{D}$; otherwise it returns `False` for Boolean attributes, and a default or a null value $(\perp)$ for non-Boolean attributes.

Finally, we resort to three-valued logic, with three possible values (i.e., $true$, $false$, $unknown$), where $unknown$ is the interpretation of the unstable value $(\varnothing)$. We do not consider the null value $(\perp)$ as unstable and we do not permit the null value for Boolean variables. We define the evaluation of any logical binary or unary operator involving `Unknown` as `Unknown`, whereas we rely on traditional two-valued logic when no *unknown* value is present. Also, when dealing with different system snapshots Σ, to differentiate an attribute value among different snapshots, when it is not clear from the context, we extend the definition τ to include Σ as input parameter as follows: $\tau_k(\Sigma, attr, \rho_k)$ or $\tau(\Sigma, attr)$.

δ is the subscription's consumption policy that describes how the internal state of a subscription changes after consuming an event. The consumption policy is explicitly defined in Section 4.4.

$N(\rho_k)$ is the subscription's notification policy that is also a disjunction over $\nu_i(\rho_k) \in N(\rho_k)$ and instantiated and correlated with each external event ρ_k. The notification consists of the notification schema that describes the content of the

event(s) (its payload) and a set of conditions $\nu_i(\rho_k)$ that dictate how the content of the event is generated.

$\Psi(\rho_k)$ defines the relationship between a subscription's matching condition Φ and a subscription's notification policy N and is represented as a set of ordered pairs $\langle \phi \in \Phi, \nu \in N \rangle$, where each ϕ_i is associated with the corresponding ν_i, meaning, when the matching condition ϕ_i is satisfied, then the notification condition ν_i is evaluated:

$$\Psi_s(\rho_k) = \bigcup_{\phi_i \in \Phi} \langle \phi_i(\rho_k), \nu_i(\rho_k) \rangle \tag{4.2.4}$$

An instance of the subscription $\mathcal{S}$ consists of an internal state $\Sigma_j^{\mathcal{S}}$ over the data model $\mathcal{D}$. The internal state of a subscription can only be changed upon receiving (consuming) an external event or generating an event (notification). In general, the internal state together with an event shapes the subscription operational semantics $\mathcal{O}_{\mathcal{S}}$ (a.k.a., the matching semantics), which is summarized as a 6-tuple:

$$\mathcal{O}_{\mathcal{S}} = \langle \Sigma_j^{\mathcal{S}}, e, t, x, \Sigma_{j+1}^{\mathcal{S}}, Gen \rangle \tag{4.2.5}$$

1. $\Sigma_j^{\mathcal{S}}$ is the current internal state of the subscription $\mathcal{S}$.
2. e is an occurrence of an external immutable event.
3. t is the logical time, which is greater than all logical timestamps occurring in $\Sigma_j^{\mathcal{S}}$.
4. x is a variable that ranges over the IDs of instances of $\mathcal{S}$. This is referred to as the context variable of $\mathcal{S}$.
5. $\Sigma_{j+1}^{\mathcal{S}}$ is the internal state after consuming event e.
6. Gen is the set of generated immutable event occurrences (generated by the notification policy) as reaction to the external event ρ_k.

Consequently, the operational semantics for a subscription $\mathcal{O}_{\mathcal{S}}$ is formally defined as follows.

Definition 2. Given a subscription $\mathcal{S}$ with internal state $\Sigma_j^{\mathcal{S}}$ and an external event e at time t for the instance x (denoted by ρ) of $\Sigma_j^{\mathcal{S}}$, the subscription $\mathcal{S}$ examines e and either accepts e and makes a transitions from $\Sigma_j^{\mathcal{S}} \xmapsto{e,t,x} \Sigma_{j+1}^{\mathcal{S}}$, or rejects e, if neither the consumption policy nor the notification policy define a state change for e.

We now formally define the publish/subscribe schema Π as follows:

$$\Pi = \langle \mathbb{P}, \mathbb{S}, \mathcal{E}, \mathcal{C} \rangle \tag{4.2.6}$$

1. $\mathbb{P}$ is a set of publications.
2. $\mathbb{S}$ is a set of subscriptions.
3. $\mathcal{E}$ is the event schema that captures both publications' event and subscriptions' notification schemes.
4. $\mathcal{C}$ is the communication state maintaining for each external event its type (e), the logical time of its occurrence (t), the subscription instance that processed it (x), and the subscription type $\mathcal{S}$, formalized as

$$\mathcal{C} = \langle \mathtt{e}, \mathtt{t}, \mathtt{x}, \mathcal{S} \rangle. \tag{4.2.7}$$

Without loss of generality, if there is more than just a single publisher for external events our formal pub/sub model assumes the following two properties:

i) Each subscription instantaneously examines the external event e according to the subscription operational semantics.

ii) At any instant in time, only a single subscription is examining e in Π.

These assumptions simplify the correctness proof for our mapping (cf. Section 4.5) as external events are inspected in the same sequential order by all subscriptions.

We refer to this as the *pseudo-serializable execution* property of Γ[1].

An instance of the publish/subscribe schema Π is defined as a sequence of global snapshots of $\mathbf{\Sigma}_1 \cdots \mathbf{\Sigma}_k$ over a discrete time space t, where $\mathbf{\Sigma}_i = \{\Sigma_k^{\mathcal{C}}, \Sigma_j^{\mathcal{S}}\}$, $\Sigma_k^{\mathcal{C}}$ is the communication state at time t over Π, and $\Sigma_j^{\mathcal{S}}$ is the internal state of each subscription instance of $\mathcal{S}$. Π's operational semantics is summarized as follows:

$$\mathcal{O}_\Pi = \langle \mathbf{\Sigma}_k, e, t, x, \mathbf{\Sigma}_{k+1} \rangle \tag{4.2.8}$$

Here, $\mathbf{\Sigma}_k$ is the current snapshot of Π. And e is an occurrence of an external event that is *pending*, implying that there is at least one instance x of at least one subscription $\mathcal{S}$ that has not yet examined e at logical time t. $\mathbf{\Sigma}_{k+1}$ is the new global snapshot of Π. Formally, the operational semantics of the pub/sub model $\mathcal{O}_\Pi$ is defined as follows:

Definition 3. Given a pending event $\rho_k = (e, t, x)$ and a subscription $\mathcal{S}$ that has yet to examine ρ_k and having the current state $\Sigma_j^{\mathcal{S}}$, then the global snapshot advances instantaneously from $\mathbf{\Sigma}_k \xmapsto{e,t,x} \mathbf{\Sigma}_{k+1}$, namely:

1. The communication state transitions from $\Sigma_k^{\mathcal{C}} \xmapsto{e,t,x} \Sigma_{k+1}^{\mathcal{C}}$, i.e., event e was sent to $\mathcal{S}$ for instance x.

2. The subscription $\mathcal{S}$ examines the external event e in accordance to $\mathcal{O}_{\mathcal{S}}$; hence, $\mathcal{S}$ either accepts e and transitions from $\Sigma_j^{\mathcal{S}} \xmapsto{e,t,x} \Sigma_{j+1}^{\mathcal{S}}$, or rejects e.

We define a valid execution sequence over Π as one that corresponds to a *pseudo-serializable execution* such that at any instant in time, Π transitions only once from state $\Sigma_k^{\mathcal{C}} \xmapsto{e,t,x} \Sigma_{k+1}^{\mathcal{C}}$, and only a single instance of subscription $\mathcal{S}$ receives an event e and transitions from $\Sigma_j^{\mathcal{S}} \xmapsto{e,t,x} \Sigma_{j+1}^{\mathcal{S}}$ (if necessary). Notably, at any instant in time, many subscriptions (or many instances of a single subscription) may be waiting to receive the event e; however, the *pseudo-serializable execution property*

[1]The practical implication of these assumptions is that with multiple external event publishers, all external events must be serialized. This requires a synchronization mechanism between external event publishers in order to generate a total order over a discrete timespace t. A solution to this problem in content-based publish/subscribe systems is presented in [105]

does not impose any restriction on the order in which subscriptions (or instances of a subscription) must receive the event e. Therefore, any non-deterministic selection of subscriptions (or instances), that results in an instantaneous examination of event e at time t by a single subscription instance x, suffices. Most importantly, this pseudo-serialization requirement can be dropped when there is a single publisher of external events (cf. assumptions on the formal pub/sub model).

An event is pending only if at least one subscription instance has not examined it yet, and (in theory) every subscription instance must examine every event exactly once. Therefore, from the communication state $\mathcal{C}$, it can be inferred, which events have been processed for which instances of subscription $\mathcal{S}$ and which events are pending for which instances of $\mathcal{S}$.

Finally, in general, with more than one publisher of external events, any valid implementation of Π must guarantee the pseudo-serializable execution property.

4.3 Workflow mapping overview

Given a data-centric workflow model $\Gamma = \langle \mathcal{I}, \mathcal{R} \rangle$, we construct a pub/sub schema $\Pi = \langle \mathbb{P}, \mathbb{S}, \mathcal{E}, \mathcal{C} \rangle$ by applying a mapping function $\mathcal{M}$ such that $\mathcal{M} : \Gamma \longrightarrow \Pi$. The set $\mathbb{P}$ in our mapping consists of a single publisher which simply publishes the external events coming from the environment. However, constructing the set of necessary subscriptions, $\mathbb{S}$, is more subtle and is primarily derived from the set of PAC rules and the PDG for a given model, Γ. In addition, we require a set of subscriptions for bookkeeping purposes such as updating data and status attributes and determining the start and the end of a B-step.

We define subscriptions both for processing relevant PAC rules and maintaining the current values for status and data attributes. In general, two classes of subscriptions arise: (1) *Application-specific* subscriptions which capture the core of the workflow operational semantics encoding both the PAC rule semantics and the PDG topological sort order semantics. (2) *Generic* subscriptions which implement a bookkeeping mechanism to provide a consistent view of the data with an implicit locking

mechanism. This mechanism maintains multiple versions of values for all attributes from the information model and applies updates in a deterministic order dictated by the order of external events. Hence, there is one version for each ρ_k, i.e., for each B-step. These two classes of subscriptions also incorporate the time semantics of the workflow meta-model, which is based on the external event received from the single publisher in our publish/subscribe formulation. Therefore, subscriptions are event-relativized in a sense that each subscription evaluates its conditions, implements its consumption policy, and sends its notification in the context of each external event in isolation, which forms a B-step.

In our mapping, $\mathcal{M} : \Gamma \longrightarrow \Pi$, we require the following set of subscriptions for key workflow operations: [$\mathcal{S}_{\oplus s}$] and [$\mathcal{S}_{\ominus s}$] for satisfying/falsifying or validating/invalidating status attributes s; [$\mathcal{S}_s$] for updating the status attribute s; [$\mathcal{S}_d$] for updating the data attribute d; [$\mathcal{S}_{source}$] for identifying the start of a B-step; and [$\mathcal{S}_{sink}$] for identifying the end of a B-step, where the $\oplus$ or $\ominus$ polarity, denotes a positive or a negative change in status attributes.

Next, we provide a high-level overview of each subscription. The high-level representation and interaction among subscriptions (represented as oval) is also depicted in Figure 4.3.1. The directed, solid arrows indicate the flow of events among subscriptions and the (bright-colored) directed, dashed arrows indicate events received from and sent to the environment, while the (black) dashed lines are bookkeeping messages for maintaining a consistent view of the attributes. What is not shown in the figure, for improved readability, is that there must be an arrow from every node to the node $\mathcal{S}_{sink}$ to determine the end of a B-step.

The precise meaning of the arrows becomes evident in Section 4.4, after formally defining each subscription.

[$\mathcal{S}_{\oplus s}$], [$\mathcal{S}_{\ominus s}$]: For each status attribute s in the information model of Γ, $\mathcal{I}$, we add the subscription $\mathcal{S}_{\oplus s}$, for validating the attribute s and the subscription $\mathcal{S}_{\ominus s}$ for invalidating s. The subscription's condition Φ is derived based on the PAC rules' prerequisite and antecedent conditions. Hence, Φ is an application-specific condition.

[$\mathcal{S}_s$]: For each status attribute s in $\mathcal{I}$, we add the subscription $\mathcal{S}_s$ that listens to

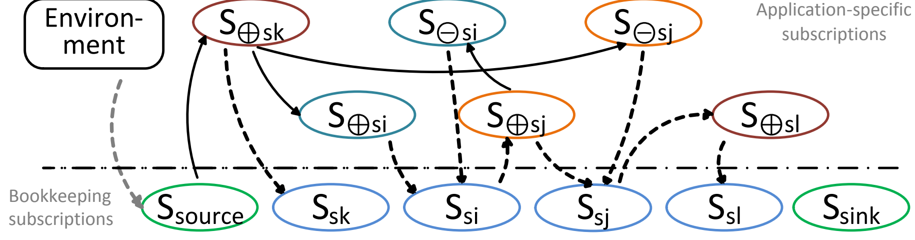


Figure 4.3.1: High-level illustration of subscription flow.

updates (i.e., notifications of $\mathcal{S}_{\oplus s}$ and $\mathcal{S}_{\ominus s}$) for s. Hence, $\mathcal{S}_s$'s Φ is a generic condition.

[$\boldsymbol{\mathcal{S}_d}$]: For each data attribute d in $\mathcal{I}$, we add the subscription $\mathcal{S}_d$ that listens to updates on d at the outset of the B-step. Hence, $\mathcal{S}_d$'s Φ is also a generic condition.

[$\boldsymbol{\mathcal{S}_{source}, \mathcal{S}_{sink}}$]: For identifying the beginning and ending of a B-step we add the source subscription $\mathcal{S}_{source}$ and the sink subscription $\mathcal{S}_{sink}$, respectively. Both subscriptions are intended for bookkeeping purposes. Thus, their conditions are also generic.

4.4 Mapping formalization

The subscription plays a central role in formulating the mapping of a data-centric workflow model, $\Gamma = \langle \mathcal{I}, \mathcal{R} \rangle$, into the pub/sub abstraction given by $\Pi = \langle \mathbb{P}, \mathbb{S}, \mathcal{E}, \mathcal{C} \rangle$. We formalize the semantics of a subscription $\mathcal{S} \in \mathbb{S}$ as described in Equation 4.2.2, where its condition $\Phi(\rho_k)$, consumption policy $\delta(\rho_k)$, and notification policy $N(\rho_k)$ are instantiated and associated with an external event ρ_k, and $\Psi(\rho_k)$ interrelates condition and corresponding notification.

4.4.1 Matching and notification policies

In this section, we start by providing a detailed account of the mapping of the workflow's application-specific semantics, namely, encoding of PAC rules and the

PDG topological sort order, into a set of subscriptions. In addition, we provide the foundation for a mapping that emulates the generic-execution semantics including the necessary bookkeeping mechanism as a set of subscriptions. We guarantee that the workflow correctly executes by ensuring data consistency and the B-step semantics.

PAC rules and PDG mapping

We first define the Γ application-specific conditions for subscriptions $\mathcal{S} \in \mathbb{S}$. Each logical formula $\phi_i(\rho_k) \in \Phi$ is defined as follows:

$$\phi_i(\rho_k) = \overbrace{\psi_{i,PDG}(\rho_k)}^{\text{PDG Predecessors}} \;\wedge\; \overbrace{\psi_{i,PseudoClock}(\rho_k)}^{\text{Event-based Pseudo Clock}} \tag{4.4.1}$$

Here, $\psi_{i,PDG}$ is the PDG predecessor component, that is, a logical formula that encodes the PDG topological sort order, i.e., $\psi_{i,PDG}$ is a logical formula that evaluates to true when all variables in $\mathcal{D}$ have stabilized (cf. detailed descriptions in the paragraphs below). $\psi_{i,PseudoClock}$ is a logical formula that enforces that subscriptions are processed based on the order of external events, i.e., it guarantees event-order serialization. The second component of Ψ, the notification expression, $\nu_i(\rho_k) \in N$, is defined as follows:

$$\nu_i(\rho_k) = \begin{cases} \gamma_{\rho_k}, \mathcal{S}^{visited}_{s_{\rho_k}} & \text{if } \psi_{i,SAT}(\rho_k) \\ \overline{\gamma}_{\rho_k}, \mathcal{S}^{visited}_{s_{\rho_k}} & \text{if } \forall \nu_i \in N, \neg(\psi_{i,SAT}(\rho_k)) \\ \texttt{WAIT} & \text{if } \exists \phi_i \in \Psi_i, \neg(\phi_i) \end{cases} \tag{4.4.2}$$

Here, $\mathcal{S}^{visited}_{s_{\rho_k}}$ is an event that indicates that the subscription $\mathcal{S}$ was successfully visited for the external event ρ_k , i.e., (partially) completed as defined in Section 4.4.2. And, γ_{ρ_k} is an event that represents the consequent of the PAC rule indicating a change (either a positive or a negative) to a particular status attribute $s \in \mathcal{I}$ ($\gamma = \odot s$), while $\overline{\gamma}$ indicates no change to status attribute s. In addition, each event γ_{ρ_k} and $\overline{\gamma}_{\rho_k}$ contains the current value of the status attribute s in the context of the external event ρ_k. $\psi_{i,SAT}(\rho_k)$ is a logical formula derived from a PAC rule's prerequisite π

and antecedent α and `WAIT` is an indicator that implies that not all subscription conditions (ϕ_i) have been satisfied. The notification policy is explained in detail in a separate paragraph below.

Application-specific condition To construct the application-specific condition, we adapt the PDG construction algorithm [37], which operates on the set of PAC rules $\mathcal{R}$. Suppose each PAC rule has the form $\langle \pi, \alpha, \gamma \rangle$ that stands for prerequisite, antecedent, and consequent, respectively. Then, the antecedent α of a PAC rule is constructed according to the template **on** $\xi(x)$ **if** $\varphi(x)$, where each expression *expr* $\in \xi(x)$ is of the form on-event *onEventType* or $\odot s$, where *onEventType* indicates requiring an external event of the type given by *onEventType* and $\odot$ means waiting for a positive, $\oplus$, or negative, $\ominus$, change in status attribute s. Similarly, every expression *expr* $\in \gamma$ also follows the form $\odot s$. However, expression *expr* $\in \varphi(x)$ has the form s, which simply indicates a stable value for status attribute s. A value is stable if it no longer changes in the current B-step.

We first collapse instances of PAC rules, $R_i \in \mathcal{R}$, that have identical π and γ into a *super PAC rule* given by $\langle \pi, \mathcal{A}, \gamma \rangle$, where $\mathcal{A} = (\vee_{\alpha \in R_i} \alpha)$. In general, PAC rules share identical π and γ because for a given status attribute s there exist multiple rules that satisfy or falsify it. The notion of a super PAC rule simplifies the mapping. Therefore, the (super) PAC rule R is mapped to $\mathcal{S}_{\odot s}$, where $\odot s \in \gamma$.

Example 1 In our example workflow, there are three PAC rules that share the common prerequisite $\pi =$ `PE : pp` and consequent $\gamma = \ominus$`PE : pp` (cf. Table 4.1.4). These PAC rules represent the incoming edges in the PDG for node `(-,PE:pp)` (cf. Figure 4.1.2). Hence, they can be collapsed into the super PAC rule $\text{PAC}_{\ominus\text{PE:pp}}$ that represents the invalidation of milestone `PE:pp`.

π	$\mathcal{A}$	γ
PE:pp	($\ominus$ ED:cp) $\vee$ ($\oplus$ ED:cp $\wedge$ LR) $\vee$ (latestIncEvent = “R:RedoExportDocs” $\wedge$ LR)	$\ominus$ PE:pp

The super PAC rule is mapped to subscription $\mathcal{S}_{\ominus\texttt{PE:pp}}$.

The relation $\Psi_{\odot s} \in \mathcal{S}_{\odot s}$ (an application-specific condition and notification) is constructed through various mapping stages, which are described next.

Each PAC rule is used to construct the subscription's matching condition, $\langle \pi, \mathcal{A}, \gamma \rangle \in \mathcal{R} \rightsquigarrow \Phi \in \mathcal{S}_{\odot s}$. More specifically, we derive each $\phi_i \in \Phi$ based on the PAC rule as follows:

$$\mathcal{M}_{\Phi} : \alpha_i \in \mathcal{A} \longrightarrow \phi_i \in \Phi. \tag{4.4.3}$$

Intuitively speaking, the antecedent of a PAC rule (α) forms the matching condition (Φ). In case of a super PAC rule, each antecedent of the original PAC rules that are collapsed within this super PAC rule ($\alpha_i \in \mathcal{A}$) forms a single component of the matching condition ($\phi_i \in \Phi$).

PDG predecessors The key component of ϕ_i, denoted by $\psi_{i,PDG} \in \phi_i$, is at the core of the subscription mapping and incorporates the notion of PDG predecessors, an integral part of encoding the PDG topological sort order semantics of Γ into Π. This mapping stage is represented by:

$$\mathcal{M}_{\psi_{i,PDG}} : \alpha_i \in \mathcal{A} \longrightarrow \psi_{i,PDG} \in \phi_i. \tag{4.4.4}$$

Thus, for each $\phi_i \in \Phi$, we construct $\psi_{i,PDG} \in \phi_i$ from the corresponding $\alpha_i \in \mathcal{R}$. The actual definition of $\psi_{i,PDG}$ is derived by adapting the PDG construction algorithm that examines the antecedent component of each PAC rule identifying the set of status attributes whose values must stabilize before firing a PAC rule, i.e., before evaluating the subscription. We formally define $\psi_{i,PDG}$ as a set of on-events that listen for positive or negative change in variables appearing in the PAC rule's antecedent, which encodes the three different forms a sentry can have:

$$\psi_{i,PDG}(\rho_k) = \bigwedge_{\odot s \in \xi(x)} \tau_k(\texttt{on}\odot\texttt{s}, \rho_k) \ \wedge \bigwedge_{s \in \varphi(x)} \tau_k(\texttt{on.s}, \rho_k), \tag{4.4.5}$$

where `on⊙s` refers to events that report a change or no change to s (i.e., the **on** $\xi(x)$ component in antecedent α). and `on.s` refers to an event that holds the current value of s (i.e., the **if** $\varphi(x)$ component in α).

Event-based pseudo clock The second component of ϕ_i is $\psi_{PseudoClock}$, which enforces that each subscription is processed, namely, its condition ϕ_i is satisfied, in the order in which external events arrive. Therefore, external events act as a pseudo clock. The operation of this pseudo clock is defined by a logical formula as follows:

$$\begin{aligned}\psi_{PseudoClock}(\rho_k) = (\nexists \rho_j, \rho_j \in \Sigma^{\mathcal{S}},\\ \neg(\tau_j(\texttt{isVisited}, \rho_j))\wedge\\ \tau_j(\texttt{eventTime}, \rho_j) < \tau_k(\texttt{eventTime}, \rho_k)\wedge\\ \tau_j(\texttt{subInstance}, \rho_j) = \tau_k(\texttt{subInstance}, \rho_k)).\end{aligned} \tag{4.4.6}$$

Example 1—cont. As $\text{PAC}_{\ominus\text{PE:pp}}$ is originally comprised of three individual PAC rules, the subscription condition Φ contains three disjuncts representing the original antecedents (i.e., ϕ_1, ϕ_2, and ϕ_3), which results in the following PDG predecessor components:

$$\begin{aligned}\psi_{1,PDG}(\rho_k) &= \tau_k(\texttt{onED : cp}, \rho_\texttt{k})\\ \psi_{2,PDG}(\rho_k) &= \tau_k(\texttt{onED : cp}, \rho_\texttt{k}) \wedge \tau_k(\texttt{on.LR})\\ \psi_{3,PDG}(\rho_k) &= \tau_k(\texttt{LR}).\end{aligned}$$

Note that the external request event in PAC Rule 3 (i.e., `R:RedoExportDocs`) is not evaluated within the matching condition (here $\psi_{3,PDG}(\rho_k)$) but later on within the notification condition. The data model $\mathcal{D}$ for this subscription is as follows:

e	t	x	onED:cp	ED:cp	PE:pp	LR	visited

The complete subscription condition Φ is then:

$$\Phi = \phi_1 \vee \phi_2 \vee \phi_3$$
$$\Phi = (\psi_{1,PDG}(\rho_k) \vee \psi_{2,PDG}(\rho_k) \vee \psi_{3,PDG}(\rho_k)) \wedge \psi_{PseudoClock}(\rho_k)$$

Application-specific notification Once the PDG requirement (i.e., $\psi_{i,PDG} \in \phi_i$) for a subscription instance is satisfied, namely, all variables in α have stabilized, and all prior external events have been processed, (i.e., ($\psi_{PseudoClock} \in \phi_i$)), the corresponding notification of (ϕ_i, ν_i) is triggered. Each $\nu_i \in N$ is partially derived from the corresponding PAC rule of the super PAC rule $\langle \pi, \alpha_i, \gamma \rangle$ in accordance to Equation 4.4.2:

$$\mathcal{M}_N : (\pi, \alpha_i \in \mathcal{A}, \gamma) \in \mathcal{R} \longrightarrow \nu_i \in N. \tag{4.4.7}$$

The key component of ν_i is a logical formula $\psi_{i,SAT}$, which describes the behavior of the notification policy. Before, giving the definition of the logical formula $\psi_{i,SAT}$, we must re-write π and α_i. This re-writing is necessary for abiding by the workflow semantics, in which each variable in π must use its last recent value from the last completed B-step (if any), while each variable in α_i must use its most recent value. Thus, we re-write π, which consists of only Boolean variables, as follows:

$$\mathcal{M}_\pi : s_i \in \pi \longrightarrow \tau_{k-1}(\mathtt{s_i}, \rho_k). \tag{4.4.8}$$

Similarly, we re-write α, which consists of both status and data attributes, based on the most recent values as follows. $\mathcal{M}_{\alpha_i}$ consists of three stages of re-writing given by $\mathcal{M}_{\varphi\in\alpha_i}$, $\mathcal{M}^1_{\xi\in\alpha_i}$, and $\mathcal{M}^2_{\xi\in\alpha_i}$:

$$\begin{aligned}
&\mathcal{M}_{\varphi\in\alpha_i} : a_i \in \varphi \longrightarrow \tau_k(\mathtt{a_i}, \rho_k),\\
&\mathcal{M}^1_{\xi\in\alpha_i} : \odot s \in \xi \longrightarrow \tau_k(\mathtt{ons}, \rho_k) = \hat{\odot},\\
&\mathcal{M}^2_{\xi\in\alpha_i} : onEvent \in \xi \longrightarrow \tau_k(\mathtt{eventType}, \rho_k) = onEvent,
\end{aligned} \tag{4.4.9}$$

where $\hat{\odot} \in$ Boolean $\times$ Boolean refers to the type of transition for status attribute s as indicated by $\odot s$. The mapping of a PAC rule to $\psi_{i,SAT}$ is expressed as

$$\mathcal{M}_{\psi_{i,SAT}} : (\pi, \alpha_i) \in R \longrightarrow \psi_{i,SAT} \in \phi_i, \tag{4.4.10}$$

where $\psi_{i,SAT}$ is simply derived by conjunction of re-written π and α:

$$\psi_{i,SAT}(\rho_k) = \mathcal{M}_\pi \wedge \mathcal{M}_{\alpha_i}. \tag{4.4.11}$$

Example 1—cont. Based on the super PAC rule, $PAC_{\ominus PE:pp}$, we now derive the notification condition for the example in a similar fashion:

$$\begin{aligned}
\psi_{1,SAT}(\rho_k) =& \tau_{k-1}(\texttt{PE}:\texttt{pp},\rho_k) \wedge \tau_k(\texttt{onED}:\texttt{cp},\rho_k) = (\text{true},\text{false}) \\
\psi_{2,SAT}(\rho_k) =& \tau_{k-1}(\texttt{PE}:\texttt{pp},\rho_k) \wedge \tau_k(\texttt{LR},\rho_k) \wedge \tau_k(\texttt{onED}:\texttt{cp},\rho_k) = (\text{false},\text{true}) \\
\psi_{3,SAT}(\rho_k) =& \tau_{k-1}(\texttt{PE}:\texttt{pp},\rho_k) \wedge \tau_k(\texttt{LR},\rho_k) \\
& \wedge \tau_k(\texttt{eventType},\rho_k) = '\mathbf{R}:\texttt{RedoExportDocs}'
\end{aligned}$$

The components of the notification condition N (i.e., ν_1, ν_2, and ν_3) can then be derived from Equation 4.4.2. We show this for ν_1 as follows:

$$\nu_1(\rho_k) = \begin{cases} \ominus\texttt{PE}:\texttt{pp}_{\rho_k}, \mathcal{S}^{visited}_{\ominus\texttt{PE:pp}_{\rho_k}} & \text{if } \psi_{1,SAT}(\rho_k) \\ \overline{\ominus\texttt{PE}:\texttt{pp}}_{\rho_k}, \mathcal{S}^{visited}_{\ominus\texttt{PE:pp}_{\rho_k}} & \text{if } \forall\nu_i, \neg(\psi_{i,SAT}(\rho_k)) \\ \texttt{WAIT} & \text{if } \exists\phi_i \in \Psi_i, \neg(\phi_i) \end{cases}$$

Data consistency & semantics simulation

Now that we demonstrated the mapping to translate PAC rules into a set of subscriptions, next, we derive the subscriptions required for bookkeeping of status and data attributes, and the execution of Γ.

Given the relation $\Psi_{i,s}(\rho_k) = \langle\phi_i(\rho_k), \nu_i(\rho_k)\rangle$, then the generic condition ϕ_i is defined by:

$$\phi_i(\rho_k) = \tau_k(\odot a, \rho_k), \tag{4.4.12}$$

where ϕ_i essentially captures the interest in any attempt to alter the value of attribute a. On the other hand, the notification policy ν_i is expressed as follows:

$$\nu_i = a_{\rho_k}, \mathcal{S}^{visited}_{a_{\rho_k}}, \tag{4.4.13}$$

where a_{ρ_k} broadcasts the current value of attribute a and $\mathcal{S}^{visited}_{a_{\rho_k}}$ indicates that the bookkeeping subscription for a was visited for external event ρ_k.

Status attribute consistency We start with the workflow's data consistency requirement that ensures a consistent view of status attributes. We must ensure that when a status attribute changes, no race condition for updating the value arises and that every interested subscription has the most up-to-date values for its status attributes. To achieve data consistency, we add to the subscription $\mathcal{S}_s$, a generic condition, for every status attribute s in the information model of Γ, which acts as a single gateway for changing the value of s and subsequently broadcasting the final stable value of s to all interested subscriptions. The relation $\Psi_s \in \mathcal{S}_s$ is given by:

$$\phi_s(\rho_k) = \tau_k(\texttt{on}\odot\texttt{s}, \rho_k) \tag{4.4.14}$$

$$\nu_s(\rho_k) = \begin{cases} \tau_k(\texttt{s}, \rho_k) \Leftarrow \texttt{True}, \mathcal{S}^{visited}_{s_{\rho_k}} & \text{if } \tau_k(\texttt{ons}, \rho_k) = (\text{false}, \text{true}) \\ \tau_k(\texttt{s}, \rho_k) \Leftarrow \texttt{False}, \mathcal{S}^{visited}_{s_{\rho_k}} & \text{if } \tau_k(\texttt{ons}, \rho_k) = (\text{true}, \text{false}) \\ \tau_k(\texttt{s}, \rho_k) \Leftarrow \tau_{k-1}(\texttt{s}, \rho_k), \mathcal{S}^{visited}_{s_{\rho_k}} & \text{otherwise,} \end{cases} \tag{4.4.15}$$

where $\Leftarrow$ indicates assignment of the value of the right-side to the variable on the left-side.

Example 2 We now show the subscription for capturing updates on status attribute `PE:pp`. The subscription condition Φ is given by:

$$\phi_{\texttt{PE:pp}}(\rho_k) = \tau_k(\texttt{onPE : pp}, \rho_k)$$

Notification condition $N = \nu_{\texttt{PE:pp}}(\rho_k)$ is given by:

$$\tau_k(\texttt{PE : pp}, \rho_k) \Leftarrow \texttt{True}, \mathcal{S}^{visited}_{\texttt{PE:pp}_{\rho_k}} \text{ if } \tau_k(\texttt{onPE : pp}, \rho_k) = (\text{false}, \text{true})$$

$$\tau_k(\texttt{PE : pp}, \rho_k) \Leftarrow \texttt{False}, \mathcal{S}^{visited}_{\texttt{PE:pp}_{\rho_k}} \text{ if } \tau_k(\texttt{onPE : pp}, \rho_k) = (\text{true}, \text{false})$$

$$\tau_k(\texttt{PE : pp}, \rho_k) \Leftarrow \tau_{k-1}(\texttt{PE : pp}, \rho_k), \mathcal{S}^{visited}_{\texttt{PE:pp}_{\rho_k}}, \text{ otherwise}$$

Data attribute consistency Likewise, we construct a set of subscriptions that listens to events containing values for each data attribute. Upon consuming an external event, if the value in the event payload is different from the current value, then the subscription S_d generates the value, derived from the change or no change events, accordingly, as follows:

$$\phi_d(\rho_k) = \tau_k(\texttt{on}\Delta_{\texttt{e}_{\rho_k}}, \rho_k) \tag{4.4.16}$$

$$\nu_d(\rho_k) = \begin{cases} \tau_k(\texttt{d}, \rho_k) \Leftarrow d, \mathcal{S}^{visited}_{d_{\rho_k}} & \text{if } {\scriptstyle d \in \Delta_{e_{\rho_k}} \wedge \atop \tau_k(\texttt{d}, \rho_k) \neq \tau_{k-1}(\texttt{d}, \rho_k)} \\ \tau_k(\texttt{d}, \rho_k) \Leftarrow \tau_{k-1}(\texttt{d}, \rho_k), \mathcal{S}^{visited}_{d_{\rho_k}} & \text{otherwise,} \end{cases} \tag{4.4.17}$$

where $\Delta_{e_{\rho_k}}$ summarizes the data attributes appearing in e.

B-Step simulation Finally, in the workflow execution, it is crucial to identify the start and end of a completed B-step. Therefore, first, we focus on the start of a new B-step, which is achieved through subscription $\mathcal{S}_{source}$. The source subscription $\mathcal{S}_{source}$ has a special property because it is the only subscription that waits upon receiving external events e from the environment. Every incoming external event in turn establishes the start of a new B-step (referred to as the B-step *deterministic-initiation property*). Acting as a single gateway, $\mathcal{S}_{source}$ assigns increasing timestamps t to all incoming events e and thereby imposes a total order on all external events. Therefore, $\mathcal{S}_{source}$ sends the events $\mathcal{S}^{visited}_{source_{\rho_k}}$ and e_{ρ_k} that is understood by all subscriptions, where its type, time, and intended subscription instances are summarized in ρ_k. Hence, the relation $\Psi_{source} = (\phi(\rho_k), \nu(\rho_k))$ is given as follows:

$$\begin{aligned} \phi_{source}(\rho_k) &= e \\ \nu_{source}(\rho_k) &= \mathcal{S}^{visited}_{source_{\rho_k}}, e_{\rho_k}, \Delta_{e_{\rho_k}} \end{aligned} \tag{4.4.18}$$

In order to guarantee the B-step deterministic-initiation property, we must add $on\mathcal{S}^{visited}_{source\rho_k}$ to all subscriptions whose $\phi_i \in \Phi$ are empty. In the same spirit, the end

of a B-step is determined by introducing $\mathcal{S}_{sink}$ that subscribes to every subscription involved in the execution in order to establish the ending of a B-step (referred to as the B-step *deterministic-completion property*). Hence, $\Psi_{sink}(\rho_k)$ is given by:

$$\begin{aligned} \phi_{sink}(\rho_k) &= \bigwedge_{\mathcal{S}_i \in \mathbb{S}'} \tau_k(\mathtt{on}\mathcal{S}^{\mathtt{visited}}_{\mathtt{i}_{\rho_k}}, \rho_k) \\ \nu_{sink}(\rho_k) &= \mathcal{S}^{visited}_{sink_{\rho_k}}, \end{aligned} \tag{4.4.19}$$

where $\mathbb{S}' = \mathbb{S} \setminus \mathcal{S}_{sink}$

4.4.2 Consumption policy

The subscriptions' conditions and notification policies define a design-time specification of the workflow semantics under our pub/sub formulation. As opposed to this, the consumption policy specifies how to update the internal state of each subscription, Σ, at runtime. The consumption policy is tightly bound to the subscription operational semantics, denoted by $\mathcal{O}_{\mathcal{S}} = (\Sigma^{\mathcal{S}}_j, e, t, x, \Sigma^{\mathcal{S}}_{j+1}, Gen)$. To precisely model the consumption policy w.r.t. $\mathcal{O}_{\mathcal{S}}$, we discuss the subscription's evolution as it goes through the various stages of its lifecycle within a B-step: *initiation*, *modification*, *completion*, *satisfaction*, *generation*, and *termination*. Each stage and its interaction with other stages is defined next and illustrated in Figure 4.4.1.

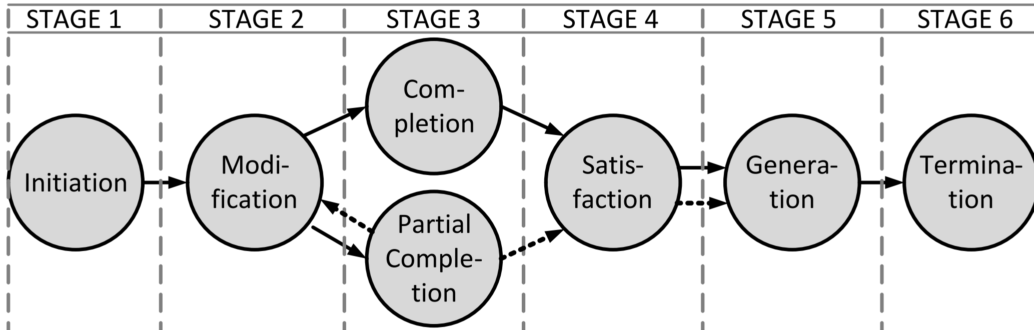


Figure 4.4.1: Consumption policy state transition.

Stage 1. Subscription *initiation* occurs for the event associated with ρ_k (within the kth B-step), when the subscription first receives the event, either directly (the event e_{ρ_k}), or indirectly (such as status or data attribute updates in the context of ρ_k). Then, `eventType`, `eventTime`, and `subscriptionIstance` are populated based on ρ_k and `isVisited` is set to false, while the rest of its attributes in $\mathcal{D}$ are set to $\varnothing$. However, if the subscription instance $x \in \rho_k$ does not exist in $\Sigma^{\mathcal{S}}$, then as part of the initialization (and creation of the new instance), all status attributes are set to false and all data attributes are set to their default values.

Stage 2. Subscription *modification* occurs for the event associated with ρ_k (within the kth B-step) after the subscription has been initiated (before or after a subscription's partial completion), when the internal state of the subscription is updated and it is transitioned according to the subscription operational semantics:

$$\mathcal{O}_{\mathcal{S}} = \Sigma_j^{\mathcal{S}} \xmapsto{E_{\rho_k} \to (e,t,x)} \Sigma_{j+1}^{\mathcal{S}}. \tag{4.4.20}$$

The internal state of the subscription changes by at most one single attribute in $\mathcal{D}$ and is characterized by the following assignment:

$$(\forall (\mathtt{a_i}, \mathtt{value}) \in E_{\rho_k}, \mathtt{a_i} \in \mathcal{D}) \to \tau_k(\mathtt{a_i}, \rho_k) \Leftarrow \mathtt{value} \tag{4.4.21}$$

Stage 3. Subscription *(partial) completion* occurs for the event associated with ρ_k (within the kth B-step) after the subscription has been initiated, when at least one of the subscription's $\phi_i(\rho_k) \in \Psi_s(\rho_k)$ has evaluated to true. If all $\phi_i(\rho_k)$ have evaluated to true, then the subscription is considered *completed*, while if at least one of $\phi_i(\rho_k)$ has evaluated to true, then the subscription is considered *partially completed*. Explicitly considering a stage for the partial completion, allows the pub/sub system to evaluate the notification policy, i.e., *Stage 4*, and generate events, i.e., *Stage 5*, before the subscription is *completed*. Hence, a tuple $\langle \phi_i(\rho_k), \nu_i(\rho_k) \rangle \in \Psi_{\mathcal{S}(\rho_k)}$ might completely evaluate to true and the corresponding notifications are generated, even if there exist conditions $\phi_j \in \Phi$ that did not yet evaluated to true. This behavior improves parallelism in execution and is indicated by the dashed lines in Figure 4.4.1.

Stage 4. Subscription *satisfaction* occurs for the event associated with ρ_k (within the kth B-step) after the subscription has been (partially) completed, when $\phi_i(\rho_k) \in \Psi(\rho_k)$ is evaluated to true, i.e., the subscription is (partially) completed, and the

subscription's corresponding notification policy, $\nu_i(\rho_k)$, evaluates to true.

Stage 5. Subscription *generation* occurs for the event associated with ρ_k (within the kth B-step) after the subscription is satisfied and when the subscription's relevant events are generated according to $\nu_i(\rho_k)$.

Stage 6. Subscription *termination* occurs for the event ρ_k (within the kth B-step) after all events have been generated by the subscription and attribute $\tau_k(\texttt{isVisited}, \rho_k)$ is assigned to true. Once `isVisited` is set to true, the tuple associated with ρ_k becomes read-only.

4.5 Workflow mapping analysis

In this section, we show that under incremental formulation (sequential execution), the data-centric workflow model Γ is equivalent to the pub/sub schema Π (distributed execution), expressed as $\mathcal{M} : \Gamma \longrightarrow \Pi$. Before establishing the correctness and equivalence of the Γ and the Π schemas, we define a set of preliminary concepts.

4.5.1 Correctness

As described in Section 4.1, the incremental operational semantics of Γ is defined as the 5-tuple $(\Sigma, e, t, \Sigma', Gen)$ and the Γ system snapshot transition, denoted by $\Sigma \overset{e}{\longmapsto} \Sigma'$, is defined as the smallest logical business step (B-step), which consists of the sequential firing of PAC rules. The B-step in its expanded form is given by $\Sigma = \Sigma_0, \Sigma_1, \Sigma_2, \cdots, \Sigma_n = \Sigma'$, where $\Sigma_0 \neq \Sigma_1$ (due to updating data attributes based on the external incoming event e) and each Σ_i is referred to as a micro-B-step. Thus, the ith micro-B-step corresponds to the firing of the ith PAC rule. Furthermore, based on the Γ semantics, each PAC rule firing results in the change of exactly one status attribute and the value of each status attribute changes at most once within a B-step (the toggle-once property). Consequently, each PAC rule is fired at most once within a B-step.

First we provide a formalization for the set of status attributes that is changed within a B-step in reaction to an external event e. Essentially, this set can be derived from the *event-relativized*-PDG.

Definition 4. The *event-relativized*-PDG for an external event e, denoted by $ePDG = (V_e, E_e)$, is a subgraph of the PDG that includes all PAC rules and their ordering that are triggered in reaction to e:

$$PDG(V,E) \supseteq ePDG = \{(V_e, E_e) | V_e \subseteq V, E_e \subseteq E\} \tag{4.5.1}$$

Definition 5. Given $ePDG = (V_e, E_e)$ for external events of type e, the *event-relativized status attribute set* for e, denoted by $\mathcal{I}_s^e$, contains all status attributes that occur in nodes V_e of $ePDG$, i.e., all status attributes that are changed within the B-step.

$$\mathcal{I}_s^e = \{s | s \in V_e,\ (V_e, E_e) = ePDG\} \tag{4.5.2}$$

Thus, the set of status attributes that is not changed within the B-step is given by $\overline{\mathcal{I}_s^e} = \mathcal{I}_s \setminus \mathcal{I}_s^e$.

We now formalize the changes in value of a status attribute over the notion of *stable attribute* values as follows.

Definition 6. A status attribute $s \in \mathcal{I}_s$ is called *stable*, denoted by $\dot{s}_\Sigma$, within a B-step caused by e iff s is within the set of attributes that are not changed as reaction to e, or s is in the event-relativized status attribute set for e and changed its value in the context of e.

$$\dot{s}_{\Sigma_i} = \begin{cases} \tau(\Sigma_{i-1}, s) \neq \tau(\Sigma_i', s) & \text{, if } s \in \mathcal{I}_s^e \\ \top & \text{, if } s \in \overline{\mathcal{I}_s^e} \end{cases} \tag{4.5.3}$$

Definition 7. We refer to initial and final system snapshot of a B-step as *complete system snapshot*, denoted by Σ (or Σ_0) and Σ' (or Σ_n), if all status attributes are stable.

$$\forall s \in \mathcal{I}_s, \dot{s}_\Sigma \tag{4.5.4}$$

Definition 8. We refer to an intermediate system snapshot within a B-step as a *partial system snapshot*, denoted by $\Sigma_i, 0 < i < n$, i.e., if not all status attributes are stable.

$$\exists s \in \mathcal{I}_s, \neg \dot{s}_\Sigma \tag{4.5.5}$$

Finally, we emphasize that the incremental formulation of the execution follows a sequential and central execution, in which the Γ semantics for the B-step execution is defined as an atomic step and each B-step consists of a finite number of micro-B-steps. Therefore, we define the concept of time in terms of a B-step such that system time advances only from t_i to t_{i+1} after processing the ith event (e_i), i.e., the completion of the ith B-step. In addition, external events are processed in the order in which they arrive—the in-order processing of external events.

Lemma 1. *The Γ incremental semantics guarantees the in-order processing of external events (when all events are published from a single source). Hence, the* B-step *execution (i.e., PAC rule firing) follows the event-order serialization.*

Proof. The proof simply follows from the Γ incremental semantics such that external events are consumed in-order and each consumed event (potentially) triggers a B-step that is executed atomically [23]. ☐

Similar to the B-step event-order serialization in the Γ semantics, the micro-B-steps within a B-step also follow a strict order which is imposed by the topological sort order of the PDG—the PDG-based serialization of micro-B-steps.

Definition 9. The Γ incremental semantics guarantees the PDG-based serialization of micro-B-steps [23].

Next, we show how the operational semantics of Γ is also guaranteed in our pub/sub formulation. As provided in Section 4.2, the pub/sub schema Π's operational semantics is also formalized as a sequence of changes in a system snapshot denoted by $\Sigma_i \xmapsto{e,t,x} \Sigma_{i+1}$, implying a single subscriber received and accepted event e.

Lemma 2. *The pub/sub operational semantics guarantees in-order delivery of events between any pair of publisher and subscriber.*

Proof. The property is a direct consequence of our pub/sub definition in Section 4.2 (cf. Definition 2). □

Corollary 1. *As consequence of Lemma 2 the mapping $\mathcal{M}$ under our pub/sub operational semantics guarantees in-order processing of external events (when all events are published from a single source).*

Furthermore, our subscription mapping for PAC rules in the Γ model processes events with respect to the order of external events (published from a single source in both Γ and Π schemas). This mapping also introduces the notion of an event-based pseudo-clock (Section 4.4) in order to achieve event-order serialization.

Lemma 3. *The mapping $\mathcal{M}$ under the pub/sub operational semantics guarantees execution of subscriptions based on event-order serialization.*

Proof. This follows from the subscription condition $\psi_{i,PseudoClock}$, which enforces that subscriptions are processed based on the order of external events. The condition $\psi_{i,PseudoClock}$ assures that a notification for event e_i is generated only if all notifications for events $e_0 \cdots e_{i-1}$ have already been generated. □

Lemma 4. *The mapping $\mathcal{M}$ under our pub/sub operational semantics guarantees the PDG-based serialization of subscriptions.*

Proof. The PDG-based serialization of a micro-B-step (single PAC rule) and a subscription (super PAC rule) is satisfied in the pub/sub semantics because the topological sort order of the PDG is directly encoded in the subscription's condition (ψ_{PDG}), which enforces that a subscription is evaluated only after all attributes have stabilized. □

With respect to the B-step execution, we also prove that the pub/sub semantics satisfy the toggle-once property.

Lemma 5. *The mapping $\mathcal{M}$ guarantees the toggle-once property of a* B-step.

Proof. The toggle-once property of a B-step, which is achieved by the PAC rule design, namely, the relationship between a PAC rule's prerequisite (π) and consequent (γ) such that, roughly speaking, $\pi \approx \neg\gamma$ and π is always evaluated w.r.t. to a system snapshot at the outset of a B-step after consuming the external event. This relation is also encoded in our subscription definition given by $\mathcal{M}_{\pi}$. □

To prove the correctness of the overall execution of the pub/sub workflow formulation, we introduce the notion of a *reachable system snapshot*: the state of the system after executing a set of external events. Therefore, the correctness of our model after processing a set of external events is determined by comparing the information model (captured by the system snapshot) of the Γ and Π schemas. If the two snapshots are identical, then our workflow to pub/sub mapping is correct, otherwise, it is incorrect.

To compare Γ and Π system snapshots, denoted by Σ^Γ and Σ^Π, respectively, we introduce two levels of equivalence, namely weak and strong equivalence. Without loss of generality, we make the following simplification in the internal data model of a system snapshot for both Σ^Γ and Σ^Π: we conceptualize Σ^Γ and Σ^Π as simply a collection of all data and status attributes given in the Γ information model. In addition, in Σ^Π, we also employ a versioning mechanism for storing this collection, in which the versioning is advanced with respect to external events. Hence, through versioning in Σ^Π, values of data and status attributes are retained separately for each external event, while in Σ^Γ, only the latest version of data and status attribute values are maintained.

Definition 10. The (partial) system snapshots Σ^Γ and Σ^Π are *weakly equivalent* up to event e_i, denoted by $\Sigma^\Gamma \Leftrightarrow_w \Sigma^\Pi$, *iff* the values of stable status attributes in both Σ^Γ and Σ^Π are equal.

$$\forall s \in \Sigma^\Gamma,\, \dot{s}_\Sigma^\Gamma \wedge \dot{s}_\Sigma^\Pi \rightarrow \tau(\Sigma^\Gamma, s) = \tau_i(\Sigma^\Pi, s) \tag{4.5.6}$$

Definition 11. The (complete) system snapshots Σ^Γ and Σ^Π are *strongly equivalent* up to event e_i, denoted by $\Sigma^\Gamma \Leftrightarrow_s \Sigma^\Pi$, *iff* all status attributes in both Σ^Γ and Σ^Π are stable and equal.

$$\forall s \in \Sigma^\Gamma,\, \dot{s}_\Sigma^\Gamma \wedge \dot{s}_\Sigma^\Pi \wedge \tau(\Sigma^\Gamma, s) = \tau_i(\Sigma^\Pi, s) \tag{4.5.7}$$

Lemma 6. *Any reachable system snapshots $\Sigma_{e_i}^\Gamma$ and $\Sigma_{e_i}^\Pi$ for event e_i are weakly equivalent.*

Proof. The only means to change data and status attributes is through external events and firing of PAC rules, respectively. Data attributes are changed through external events, since both Γ and Π semantics follow event-based serialization.

Therefore, changes to data attributes must be consistent under both formulations. The status attributes are changed through PAC rules fired within the scope of each external event; again, we showed that under both Γ and Π formulations, PAC rules follow PDG-based serialization. Moreover, the toggle-once property can be emulated under our pub/sub formulation. The toggle-once property is essential in order to avoid the infinite firing of PAC rules within a B-step, thus, achieving a finite number of micro-B-steps in a B-step. As desired, both Γ and Π result in firing of PAC rules and corresponding subscriptions derived from these PAC rules in an identical order. Hence, the values of status attribute are also guaranteed to be identical.

Moreover, the pub/sub operational semantics enables concurrent execution of external events in parallel in accordance to the PDG topological sort order. Suppose, the topological sort consists of a number of levels, where each level is associated with a set of PAC rules, i.e., subscriptions. Thus, as the external event e_i propagates through each level, all status attributes associated with visited levels will be stabilized and will be unaffected by the execution of subsequent levels of PAC rules. Therefore, granted the attribute versioning is in-place, the new external event e_{i+1} can process the subscriptions that fall in levels l_1-l_{j-1} while e_i is processing level l_j. Therefore, inductively, it can clearly be proven that as soon as one level of the topological sort order is processed by one event, the processed level is ready to accept the subsequent event. Hence, our pub/sub operational semantics is capable of processing many events in parallel while satisfying event-based and PDG-based serialization requirements. □

Lemma 7. *The time complexity of the mapping $\mathcal{M} : \Gamma \longrightarrow \Pi$ is linear w.r.t. the number of PAC rules and the size of the* Γ *model.*

Proof. We construct a set of application-specific and generic conditions by iterating over each PAC rule exactly once. In addition, we construct a generic condition for every status attribute in Γ. □

Lemma 8. *The number of subscriptions generated by the mapping $\mathcal{M} : \Gamma \longrightarrow \Pi$ is linear w.r.t. the information model $\mathcal{I}$ of the model* Γ.

Proof. For every status attribute $s \in S \subseteq \mathcal{I}$, the mapping $\mathcal{M}$ generates three subscriptions (i.e., $\mathcal{S}_{\oplus s}$, $\mathcal{S}_{\ominus s}$, and $\mathcal{S}_s$). For each data attribute $d \in D \subset \mathcal{I}$, there

is a single subscription S_d. In addition, the mapping produces the two generic subscriptions S_{source} and S_{sink}. Altogether, this results in $3 \cdot |S| + |D| + 2$, i.e., $\mathcal{O}(\mathcal{I})$ subscriptions. □

We are now in a position to prove our mapping from the workflow formulation Γ into the pub/sub abstraction Π.

Theorem 1. *The model Γ under incremental formulation is equivalent to Π in terms of the B-Step operational, which establishes the correctness of our mapping $\mathcal{M} : \Gamma \longrightarrow \Pi$.*

Proof. The necessary steps in proving the correctness for the workflow mapping in terms of the system snapshot reachability condition is summarized as follows:

(1) Event-based serialization of B-steps (firing all relevant PAC rules) and execution subscriptions (a super PAC rule) (cf. Lemmas 1,3).

(2) PDG-based serialization of a micro-B-steps (firing a single PAC rule) and execution of a subscription (a super PAC rule) (cf. Lemma 4).

(3) The toggle-once property of a B-step is satisfied in our pub/sub semantics (cf. Lemma 5).

(4) Weak equivalence of any reachable partial system snapshot $\Sigma^{\Gamma}_{e_i}$ and $\Sigma^{\Pi}_{e_i}$ for any event e_i (cf. Lemma 6) and

(5) Strong equivalence of any reachable complete system snapshot $\Sigma^{\Gamma}_{e_i}$ and $\Sigma^{\Pi}_{e_i}$ for any event e_i (cf. Lemma 6).

□

4.5.2 Overhead in B-Step execution

The number of B-steps that is executed in order to process the whole workflow depends on the characteristics of the application and is consequently not bounded

by the workflow meta-model. For a quantification of the communication costs of our mapping, we focus on the number of events generated within a single B-step, which depends on the number of PAC rules (i.e., PDG nodes) that are fired as a result of an external event e arriving (i.e., the ePDG).

In our mapping, subscriptions capture (1) PAC rule firing and (2) the bookkeeping mechanism. Once a subscription evaluates to true, events are generated and sent to other interested subscriptions (notification). Regarding (1), the number of events corresponds to all edges that are traversed in the ePDG (one event for each edge to trigger the next PAC rule) and the number of nodes visited (one event for each state change is sent to the bookkeeping subscription). As the ePDG is acyclic, the upper bound for generated events representing PAC rule firings w.r.t. the ePDG is in

$$\mathcal{O}\Big(\overbrace{|E_e|}^{\text{PAC rule fire}} + \overbrace{|V_e|}^{\text{bookkeeping}} \Big). \tag{4.5.8}$$

For (2), the estimate of generated events is based on the same argument as for (1). A bookkeeping subscription will propagate the new value of a status attribute (i.e., create an event) every time it received an update for this attribute. Within a B-step, the number of status changes is bounded by the ePDG, i.e., by the number of nodes that are possibly visited. Consequently, the upper bound for bookkeeping events, i.e. the cost for maintaining data consistency, w.r.t. the PDG, is $\mathcal{O}(|V_e|)$. Altogether, the number of events that are generated for executing a B-step is in $\mathcal{O}(|\text{ePDG}|)$.

4.6 Foundation for distribution

The mapping of the data-centric workflow model to the pub/sub schema serves to enable the robust distribution and parallel execution of each workflow element. In general, workflow elements comprise the individual tasks, transitions among them, their input/output parameters, and user roles. More specifically, in a data-centric workflow, rules are the fundamental element and capture both task invocations or transitions, respectively, and their relevant I/O parameters. In this sense, workflow

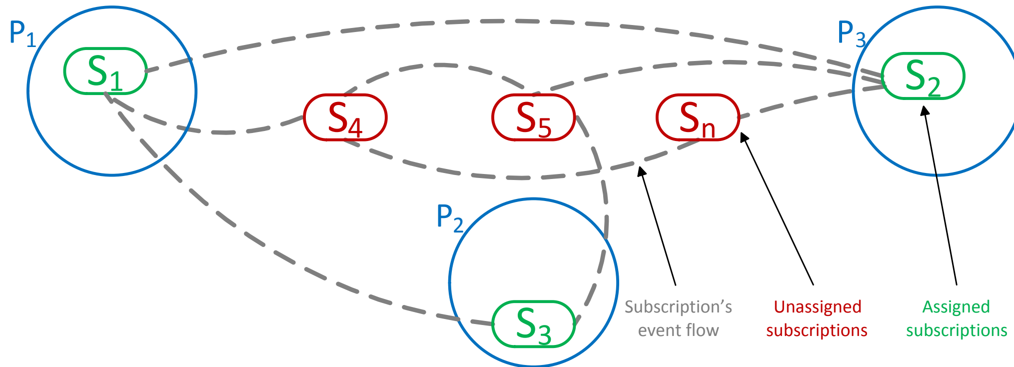


Figure 4.6.1: Illustration of subscription assignment.

distribution can be seen as the grouping of a set of rules or their mapped subscription counterparts, respectively, over a loosely coupled and distributed system. What remains unaddressed by the mapping is how to determine the actual grouping of these workflow elements in various processing sites within the pub/sub system. Yet another important property of workflow element grouping lies in the ability to easily move subscriptions among pub/sub processing nodes in order to achieve higher-level functionalities such as load balancing, replication, and availability.

One can imagine the two extreme possibilities of grouping: one that every group entails a single subscription (where each subscription is derived as was shown in Sections 4.3,4.4) or all subscriptions can be placed into a single processing site. The former approach achieves the highest level of parallelism (in a sense of distributed execution), but suffers substantially from the increased event traffic in order to coordinate and share data among elements across various processing sites. The latter approach becomes sequential (in a sense of centralized execution), but requires no event traffic for interactions among various elements.

Our goal is to lay a foundation that enables us to study the distribution of the workflow at various granularity levels in order to minimize an objective function, e.g., network traffic, while satisfying additional real-world (hard) constraints, e.g., compliance requirements: enforcing parts of a workflow to be completed in a particular geographical region, requiring that data must reside in a particular region, or following a licensing model that charges for shipping data which indirectly forces the execution to be as close as possible to the data. We formulate the workflow distribution in

terms of a portable execution unit that can be carried out in a single processing site (P-site). Thus, P-site is a processing site deployed on a given geographical location that is responsible for executing a set of elements in a given workflow such that P-site minimizes the objective function and satisfies a given set of hard constraints.

We formally define the workflow distribution problem as follow: given a set of P-sites P_i and a partial assignment of a subset of subscriptions to each of the P-site (hard constraints), then determine a complete assignment such that the network traffic among the set of P-sites is minimized. Furthermore, we require that P-sites are disjoint, namely, a single subscription cannot be assigned to more than one P-site. The solution to our problem is a complete assignment of subscription-to-P-site. Clearly, any arbitrary assignment, starting from the partial assignment, is a solution, but not necessarily one that minimizes the objective function; hence, not an optimal solution. Figure 4.6.1 illustrates an instance of our assignment problem, in which we have three P-sites, where each P-site is assigned one subscription P_1, P_2 and P_3, respectively, and we have a set of unassigned subscriptions $S_4 \cdots S_n$. Without loss of generality, we also combine subscriptions with polarity (positive and negative) into a single subscription, i.e., $S = \{S_\oplus, S_\ominus\}$.

This assignment problem is formulated as an undirected weighted graph $G = (E, V)$, where each subscription $\mathcal{S}_i$ is represented by a vertex v_i, and there is an edge between two vertices v_i and v_j *iff* the subscription $\mathcal{S}_j$ is interested in events generated by subscription $\mathcal{S}_i$ or vise versa. Also, we have a set of colors, $C = \{c_1, \cdots, c_k\}$, where each c_i corresponds to the P-site P_i. Consequently, the partial coloration (i.e., partial assignment) of the subset of vertices in G, $V' \subseteq V$, is given by the mapping function:

$$\chi : V' \longrightarrow C. \tag{4.6.1}$$

Moreover, we need a cost function to capture the communication cost between two subscriptions (relative to the size of data and protocol messages). Thus, each edge (v_i, v_j) of the graph reflects the communication cost flowing between v_i to v_j. The cost of data flow is given by:

$$\mathcal{C}_\Delta : E(G) \longrightarrow \mathbb{R}^+. \tag{4.6.2}$$

Likewise, the protocol cost is given by:

$$\mathcal{C}_\pi : E(G) \longrightarrow \mathbb{R}^+. \tag{4.6.3}$$

Finally, the total cost is given by function $\mathcal{C}_F$:

$$\mathcal{C}_F = (\mathcal{C}_\Delta + \mathcal{C}_\pi)(E(G)), \tag{4.6.4}$$

which is simply computed by summing the data and protocol cost. Therefore, under this formulation, the objective of our assignment problem is to provide a complete coloration of graph G while minimizing $\mathcal{C}_F$:

$$\overline{\chi} : V(G) \longrightarrow C, \tag{4.6.5}$$

where $\overline{\chi} = \chi$ for all $v \in V'$, such that the sum of all edge weights, given by $\mathcal{C}_F$, whose vertices are not of the same color, is minimized. Essentially, the complete graph coloration results in a complete assignment, which in turn partitions the graph into k disjoint sets of vertices such that each set is assigned to a P-site.

The intractability of our graph coloration formulation is shown by reducing the well-known *multiway cut (a.k.a., multiterminal cut) problem* [22, 98] to the graph coloration.

Definition 12. Given an undirected weighted graph $G = (E, V)$ and a set of terminals $S = \{s_1, \cdots, s_k\} \subseteq V$, then a *multiway cut* is defined as a set of edges whose removal disconnects the terminals from each other. The multiway cut asks for a minimum weight edge set whose removal disconnects the terminals.

The problem of computing the minimum weight multiway cut is NP-hard for any fixed size $k > 2$ [22]. For $k = 2$, the problem is tractable and can be solved optimally using the standard max-flow, min-cut algorithm. Furthermore, for $k \geq 3$, there exists a greedy algorithm with a $2 - \frac{2}{k}$ approximation ratio [98]. This greedy algorithm [98] consists of two phases:

1. For each $i = 1 \cdots k$, compute a minimum weight isolating cut $\mathcal{I}_i$ for each s_i. This cut is computed optimally using the max-flow algorithm by construction a new instance of the min-weight cut problem which consists of only two terminals, namely, s_i and $S - \{s_i\}$.
2. Discard the maximum weight cut $\mathcal{I}_j$ and output the union of the rest, denoted by:

$$\mathcal{I} = \Big(\bigcup_{i=1\cdots k} \mathcal{I}_i \Big) - \mathcal{I}_j.$$

   $\mathcal{I}$ disconnects any pair of terminals, hence, it is a multiway cut.

The multiway cut problem can be reduced to our graph coloration problem. We transform the multiway cut problem by assigning each terminal vertex to a different color (i.e., partial color assignment), and we assign a color to each non-terminal vertex (i.e., complete color assignment), where the color is chosen from the set of colors used for the terminal vertices, such that we minimize the edge cut between vertices of different colors. Hence, there exists a polynomial reduction of the classical multiway cut problem to our graph coloration problem (i.e., *colored multiway cut*).

Theorem 2. *The colored multiway cut problem is NP-hard and can be solved within a $2 - \frac{2}{k}$ approximation.*

Proof. The proof follows from reducing the known multiway cut problem to the colored multiway problem. □

In summary, we formalized the general workflow distribution problem over the pub/sub abstraction as the colored multiway cut problem. We showed that colored

multiway cut is intractable, but there exists a constant factor approximation for solving it. From a theoretical perspective, it is interesting to employ a more complex communication cost function in the workflow distribution which collapses all edges leaving from the subscription S_i to all interested subscriptions residing in a different processing site P-site because it is sufficient to transmit a message once from S_i to each interested subscriptions in a P-site. The collapsing of edges and the extension of the problem to a directed graph, instead of an undirected graph, leads to new challenges for future research. However, these new restriction do not affect the hardness of the problem, namely, the problem remains intractable.

4.7 Summary

In this chapter, we established the theoretical foundation for the safe distribution and the parallel execution of data-centric workflows over the loosely-coupled and distributed publish/subscribe abstraction. To this end, we developed a polynomial-time mapping of data-centric workflows into the pub/sub abstraction to achieve distributed and parallel workflow execution. We proved the correctness of our mapping through an equivalence of reachable system snapshots and we proved the hardness of the optimal workflow distribution problem over the pub/sub abstraction. Finally, we employed a greedy algorithm with a constant factor approximation for this problem.

Chapter 5

Geo-Distribution of Flexible Business Processes

In this chapter, we present a geo-distributed execution system for flexible business processes in multi-organizational scenarios. Our system relies on GSM [23], a generalization of flexible business processes, which forms the execution semantics of the Case Management Model and Notation (CMMN) [62, 73]. In GSM, a workflow is based on a global *information model* that represents both application data from all involved organizations as well as information about the current state of the process. The execution semantics are specified as a set of PAC rules that are fired on receiving an external event from the environment to evaluate the current snapshot of the GSM system and evolve it to the next state.

A distributed GSM workflow engine forms the core of our system. It supports locality of data by enabling the deployment of system components, which need to access certain data, in the IT infrastructure of the respective data owner. Our system is based on the formal mapping of GSM into the publish/subscribe abstraction described in Chapter 4. The loosely-coupled nature of pub/sub supports both the ad-hoc character of flexible business processes and the rule-based execution semantics. In contrast to the previous chapter, this chapter provides a geo-scale system architecture including an implementation, optimization, and experimental evaluation.

We introduce *Workflow Units* (WFUs) as distributable system components that communicate over a distributed publish/subscribe middleware, and manage individual attributes from the information model or execute workflow rules. A WFU subscribes to attribute changes relevant for rule evaluation or attribute access, performs the rule evaluation or data access in the context of an event from the environment, and publishes results to other interested WFUs. The WFU representation of the workflow partitions the global information model and the control flow into executable fragments. This fragmentation is the basis for a location-aware deployment.

We start by providing formalisms to model flexible business processes based on CMMN and GSM in Section 5.1. First, we introduce an example of a flexible business process from the healthcare domain under both formulations and, then, we show how CMMN models can be expressed by GSM. Next, in Section 5.2, we describe our WFU-based geo-scale execution architecture for GSM. In Section 5.3, we present two mappings of GSM into WFUs: the first is our baseline mapping (BLM), which is a practical realization of the formal mapping developed in Chapter 4. Based on the insights obtained from BLM, we also present a novel, optimized *context-aware* mapping (CAM). Details of our implementation in PADRES [41], an enterprise-grade event management system, are shown in Section 5.4. In Section 5.5, we report on results from an experimentally study, where we compared BLM and CAM w.r.t. process latency and throughput under various configurations.

5.1 Flexible business processes formalism

We now present formalisms to model flexible business processes in an executable format. We start by introducing the Case Management Model and Notation (CMMN) [73]; then, we recap the foundation of CMMN, theGuard-Stage-Milestone meta-model (GSM) [23]. Both formalisms are described by means of an example BP handling the treatment of a patient, which we refer to as the *patient care* business process. Finally, we show that GSM is a generalization of CMMN and every CMMN model can be represented in GSM—the basis for our geo-distributed workflow execution approach.

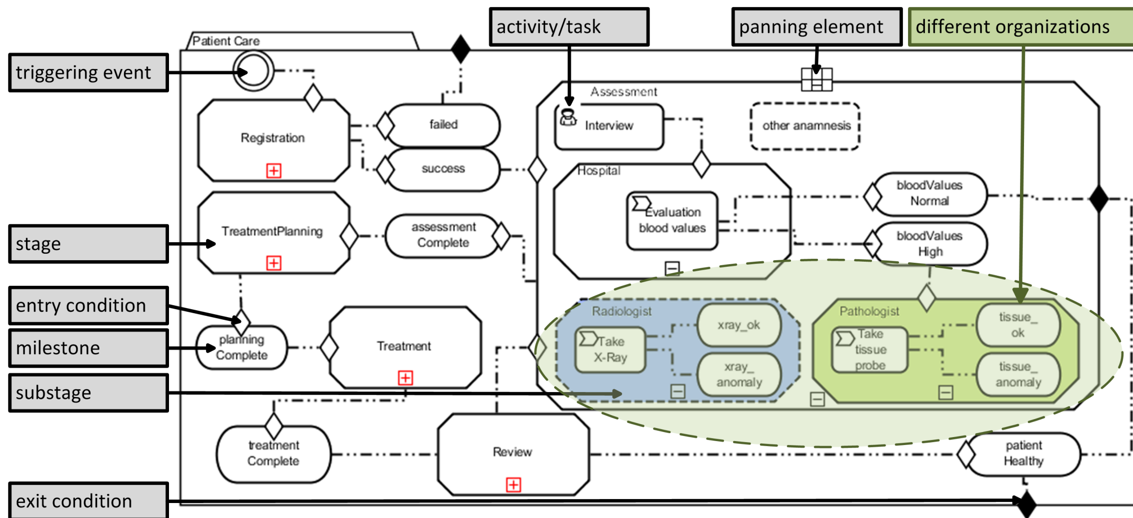


Figure 5.1.1: Patient care process in CMMN (modeled with `trisotech.com` modeler).

5.1.1 Case Management Model and Notation

Healthcare is a highly knowledge-intense domain involving many different collaborating organizations. In the following, we refer to the patient care process [93]. A corresponding CMMN model is shown in Figure 5.1.1. In CMMN, a business process is modeled as a *case* that is instantiated on a *triggering event*, e.g., the referral to a hospital. The lifecycle is modeled as a set of *stages* (octagons) clustering *substages* and/or *tasks*. A stage contains *entry conditions* (diamonds) that dictate when to open the stage. Which substages/tasks are to be executed, can be statically defined or modeled as an ad-hoc decision by annotating a stage as a *planning element* and attaching *discretionary* tasks/substages (dashed lines). Discretionary items are added on demand during runtime, offering flexibility. Stages are intended to achieve *milestones* (rounded boxes), intermediary goals in case execution, which have entry conditions to specify their achievement. Milestones are often used as entry conditions for stages. *Exit conditions*, in contrast, define when the case goal has been met and the instance can be terminated [73].

Patient care starts in stage `Registration`, where patient information is recorded. A successful registration results in achieving milestone `success`, which, in turn, opens stage `Assessment`. Here, a physician examines the patient and inquires about

symptoms. Depending on the outcome, the physician then chooses from a variety of actions required to establish a diagnosis, whereby decisions made are based on the medical knowledge of the physician. He/She might decide that additional tests are necessary or that a specialist must be involved. These alternatives are modeled as substages intended to be handled by different institutions. Referral to a `Pathologist`, for instance, requires that existing examination results should be made available to the specialist to aide in supporting a potential diagnosis. Patient data is highly sensitive and legal regulations restrict automated exchange [8]. Also, as specialists are often organized independently, they have their own data management and sub-processes, which requires an official referral. Similar to assessment, the `Treatment planning` and the actual `Treatment` are decision-intense (details are omitted in the example). After the treatment delivery, the success of the therapy is `review`ed and the patient is either discharged or the process loops back to an earlier stage.

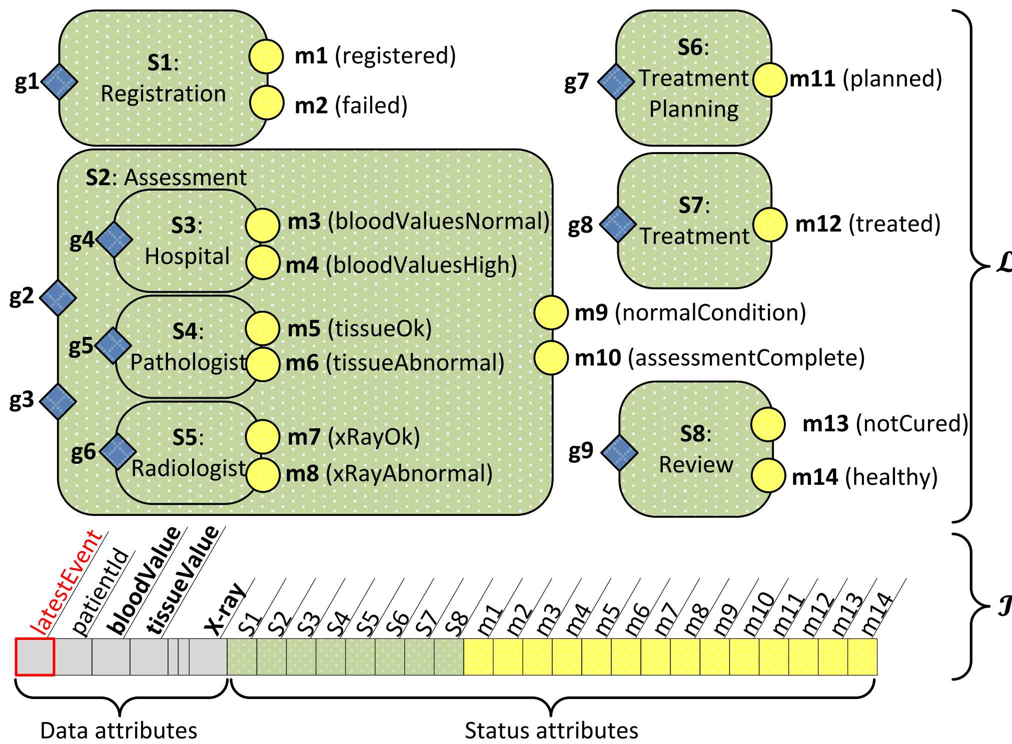


Figure 5.1.2: Patient care process in GSM.

5.1.2 Guard-Stage-Milestones meta-model

The execution semantics of the patient care CMMN model can be expressed by the Guard-Stage-Milestone meta-model (GSM) [23] as depicted in Figure 5.1.2. In GSM, a workflow is modeled as interactions of business artifacts (BA). A BA consists of a *lifecycle model*, $\mathcal{L}$, corresponding to the graphical CMMN notation, and an *information model*, $\mathcal{I}$, that captures all business relevant data (i.e., data attributes to represent application data such as patient data) and workflow-specific status information (i.e., status attributes describing the internal state of the BA). Interactions are specified in $\mathcal{L}$ and executed according to operational semantics based on declarative rules that evolve the artifact. Like CMMN, GSM provides three core elements to specify the behavior of a workflow. (1) *Milestones* are business-relevant objectives that can be achieved or invalidated (circles); (2) *Stages* (rounded rectangles) are clusters for tasks and are either *opened* or *closed*. The status attributes in $\mathcal{I}$ are Boolean attributes that capture which stages are currently opened or closed and which milestones are achieved or not. Stage opening and milestone achieving is controlled by (3) *Guards* (diamonds) that represent entry conditions and event-listeners in CMMN are specified by *sentries*. A sentry is a Boolean expression over attributes in $\mathcal{I}$.

Operational semantics The operational semantics of GSM defines the incorporation of *external events* from the business environment into the current system snapshot (i.e., an instance of $\mathcal{I}$), which corresponds to the impact on stage opening/closing, milestone achievement/invalidation, and mutation to application data [23]. Intuitively, the operational semantics of GSM is based on advancing the current system snapshot (i.e., the complete state of the workflow) into a new system snapshot after consuming exactly one external event.

Formally, the consumption of a single external event results in advancing to a new system snapshot and is encapsulated in a business step (B-Step), which is the smallest unit of business-relevant change in GSM. More specifically, in each B-Step, a set of rules, which are derived from the model, are fired according to a specific order that is derived accordingly. Each rule represents a micro-B-Step, which is an atomic data operation within a B-Step (e.g., the update of a particular milestone).

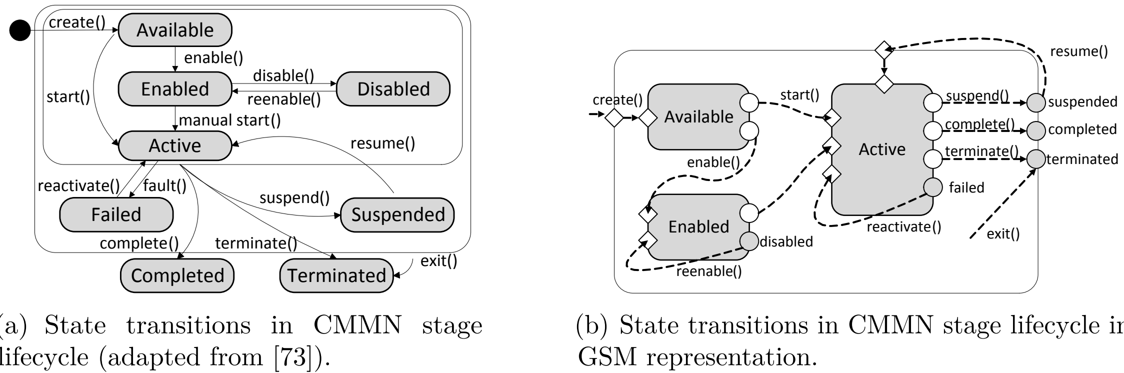
These rules are a variant of Event-Condition-Action (ECA) rules and are called *Prerequisite-Antecedent-Consequence*, or PAC rules, in GSM. Rules follow a common format and are expressed by a triple (π, α, γ). The prerequisite, π, is a Boolean expression over attribute values in $\mathcal{I}$ based on the latest B-Step. An antecedent, α, is a Boolean expression representing changes in the current B-Step such as the reception of an external event or the toggling of a milestone from `false` to `true`. The consequent, γ, describes a change to a status attribute if π and α evaluate to true. There are six templates for PAC rules (PAC-1, ..., PAC-6) for achieving/invalidating guards, for achieving/invalidating milestones and two rules for stage closing (cf. 3 for more details). Table 5.1.1 shows an excerpt from the 56 PAC rules for the patient care process depicted in Figure 5.1.2.

No.	π	α	γ
1	$\neg$S1	latestEvent = “Patient Referral”	$\oplus$ S1
2	S3	latestEvent = “T:BloodTest” $\wedge$ bloodValue > 42	$\oplus$ m4
3	S4	latestEvent = “T:TissueTest” $\wedge$ tissueValue < 1337	$\oplus$ m6
4	S2	$\oplus$ m6	$\oplus$ m10

Table 5.1.1: Excerpt of PAC Rules for “Patient care” workflow.

For instance, Rule 1 in Table 5.1.1 captures the achieving of guard `g1`, which listens for a `Patient Referral`-request to open the stage `Registration` (`S1`). The rule intuitively states that if `Registration` is currently closed (π) and there is an event of type `Patient Referral` coming from the environment (α), then open the stage (γ), i.e., toggle `S1` from `false` to `true` ($\oplus$`S1`). Similarly, Rule 3 captures the achieving of milestone `m6` (`tissueAbnormal`) once a termination event for a tissue screening containing an update to attribute `tissueValue` has been received, and the value is lower than a threshold. The achieving of `m6` ($\oplus$`m6`), in turn, represents the antecedent of a rule that captures the achieving of milestone `m10` (cf. Rule 4), which indicates that the assessment is complete. In general, PAC rules depend on other PAC rules, such that the consequent of a rule can trigger the antecedent of another rule (cf. Rules 3 and 4). This dependency is encapsulated by the *Polarized-Dependency-Graph* (PDG) [23]. When referring to a particular external event type e, the set of applicable PAC rules in reaction to e is described by the *event-relativized* PDG for e, or ePDG, for short.

Relationship between CMMN and GSM The main difference between CMMN and GSM is the lifecycle of their core elements (i.e., stages, milestones, and event



(a) State transitions in CMMN stage lifecycle (adapted from [73]).

(b) State transitions in CMMN stage lifecycle in GSM representation.

Figure 5.1.3: Lifecycle for core CMMN elements (left), GSM representation (right).

listeners). The lifecycle of a stage in CMMN has eight states and corresponding state transitions (cf. Figure 5.3(a)). In GSM, a stage lifecycle has two states (i.e., *opened* or *closed*), which might suggest that GSM is less expressive compared to CMMN.

However, we show that GSM can, in fact, be used as a meta-language to model CMMN lifcycles. The stage lifecycle in CMMN can be modeled as depicted in Figure 5.3(b). The GSM counterparts to the states in the CMMN lifecycle are highlighted in gray. Similar, the lifecycle of milestones in CMMN can be mapped to GSM. In both cases, individual states are modeled as GSM stages or milstones and the transitions are modeled based on external events (representing the transition functions). An important conclusion from this fact is that, as CMMN elements can be completely expressed by GSM, a valid GSM workflow engine is consequently also capable of executing arbitrary CMMN models. In the remainder of the chapter, we therefore focus on realizing the GSM operational semantics as these are supported by a richer theoretical foundation [23, 92].

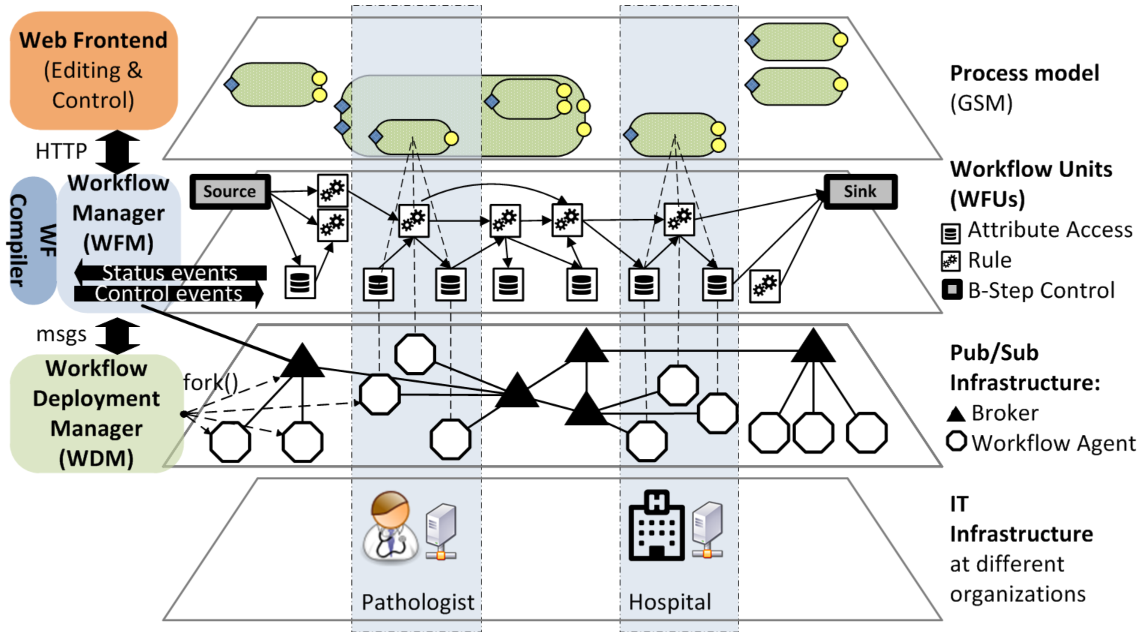


Figure 5.2.1: Distributed GSM workflow architecture.

5.2 Geo-distributed data-centric workflow architecture

An overview of our geographically-distributed workflow execution architecture is depicted in Figure 5.2.1. The basis of our architecture is a GSM representation of the workflow, $(\mathcal{R}, \mathcal{I})$, where $\mathcal{R}$ is the set of PAC rules and $\mathcal{I}$ is the information model (i.e., data and status attributes).

Business process management such as modeling, compilation, deployment, and monitoring is supported by a Workflow Manager over a Web Frontend. The workflow itself is executed by a set of independent, geo-distributable Workflow Units (WFUs), which are system components that manage PAC rules and attributes. Attributes from $\mathcal{I}$ are mapped to *Attribute Access WFUs*; every data attribute is mapped to a *Data Access WFU* (WFU_D), every status attribute is mapped to a *Status Access WFU* (WFU_S), and every PAC rule $\in \mathcal{R}$ is mapped to a *Rule WFU* (WFU_PAC). There are two additional *B-Step Control* WFUs: WFU_SOURCE handles the start of a new B-Step, i.e., the occurrence of an external event e. It acts as a global gateway

for external events that imposes a total order on B-Steps by attaching increasing timestamps t to e. WFU_{SINK} handles the termination of a B-Step by representing a gateway that checks whether all PAC rules have been evaluated and all attributes have been updated in the B-Step. We present two mappings of GSM into the WFU representation in Section 5.3. Automated workflow compilation according these mappings is provided by a Workflow Compiler that generates declarative descriptions of WFUs. Examples for such descriptions are provided in Listings 5.3.1, 5.3.2, 5.3.3, and 5.3.4, and will be discussed in detail in Section 5.3.

In the following, we rely on the standard terminology established for pub/sub to describe the internals of a WFU [41]. In pub/sub, subscribers express their interests in certain information by *subscribing* to an event filter via issueing *subscriptions*. Publishers produce events and *publish* these as *publications* to the pub/sub system. The pub/sub system matches publications and subscriptions, and *notifies* subscribers about matching events.

Every WFU is characterized by three components:

1. *subscriptions* ($\mathcal{S}$) define the interest of the WFU in updates to attributes that are relevant to evaluate the *application logic*.
2. the *application logic* ($\mathcal{AL}$) of the WFU within the context of a B-Step. The application logic is represented as a set of rules, which are evaluated once all attributes are available for the B-Step. Depending on the WFU type, the application logic either represents a PAC rule, or the access to an attribute to persist or retrieve updated values if required, or the detection of the start or end of a B-Step. The result of WFU evaluation is described by a set of *notifications*.
3. *notifications* ($\mathcal{N}$) to update other WFUs that are interested in these results. Essentially, $\mathcal{N}$ defines a set of publication templates, which a WFU instantiates and publishes depending on the result of $\mathcal{AL}$.

A WFU mapping partitions the global data-model $\mathcal{I}$ into a set of overlapping data models $\mathbb{D}_{WFU} = \{\mathcal{D}^1, \ldots, \mathcal{D}^{|WFU|}\}$, such that $\mathcal{D}^i$ contains the attributes $\in \mathcal{I}$ that

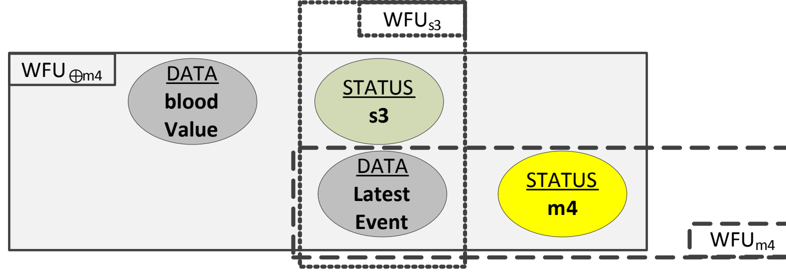


Figure 5.2.2: WFU partitioning of global data model.

WFU_i maintains and $\bigcup_{i \in |WFU|} \mathcal{D}^i = \mathcal{I}$. The set of attributes WFU_i maintains is comprised of those it subscribes to and those it updates itself by evaluating the application logic. WFU_i maintains a multi-version implementation of $\mathcal{D}^i$ containing the attribute values for all B-Steps.

Figure 5.2.2 depicts the overlapping that occurs when partitioning the global data model of the patient care process. The example is restricted to three WFUs constructed according to our mappings ($WFU_{\oplus m4}$, WFU_{m4}, and WFU_{s4}). $WFU_{\oplus m4}$ encodes the second PAC rule in Table 5.1.1. It subscribes to status attribute `s3` and data attributes `latestEvent` and `bloodValue`, and publishes status attribute `m4`. Hence, the data model is comprised of all four attributes depicted in the figure:

$$\mathcal{D}^{WFU_{\oplus m4}} = \{\text{latestEvent}, \text{bloodValue}, \text{s3}, \text{m4}\}$$

The remaining WFUs are *Status Access WFUs* for `s3` and `m4`. Compared to $WFU_{\oplus m4}$, their data models overlap with the corresponding status attribute and `latestEvent`.

$$\mathcal{D}^{WFU_{m4}} = \{\text{latestEvent}, \text{m4}\}, \mathcal{D}^{WFU_{s3}} = \{\text{latestEvent}, \text{s3}\}$$

WFUs are the basic unit of geographic distribution in our architecture. A Workflow Deployment Manager (WDM) assigns each WFU to a workflow agent (WF Agent), a lightweight system component that is capable of executing WFU declarations as provided by the compiler. WF Agents implement the pub/sub interface and are connected to a network of message brokers [41]. The broker network constitutes a pub/sub system that manages subscriptions of WFUs and routes publications among them. A concrete deployment of WFUs to WF Agents is flexible because the set of

WFUs represents the most fine-grained view of a GSM process. The possibilities range from a centralized deployment at a single machine to host WF Agents for all WFUs, to a completely decentralized deployment, where every WFU is managed by a separate WF Agent on a separate machine.

The WFU representation enables a location-aware deployment, where every *Attribute Access WFU* is deployed in the IT infrastructure of the legal owner of the data (e.g., the hospital for `bloodValue`). Similarly, if a WFU_{PAC} contains sensitive attributes to evaluate a PAC rule, it is also deployed in the IT infrastructure of the data owner. Rule 2 in Table 5.1.1, for instance, contains access to `bloodValue` in its antecedent and is deployed in the hospital infrastructure.

Workflow instances are controlled by the Workflow Manager that receives events e from the frontend, triggers the engine with e, and forwards updates back to the frontend.

5.3 GSM to Workflow Unit mappings

We now provide a detailed account of our mappings and focus on the mapping functions for the five WFU types: $\text{WFU}_{\text{SOURCE}}$, WFU_{PAC}, WFU_{S}, WFU_{D}, and WFU_{SINK}. We present the functions for both baseline (BLM) and context-aware mapping (CAM) and describe the three components, *subscriptions* ($\mathcal{S}$), *application logic* ($\mathcal{AL}$), and *notifications* ($\mathcal{N}$). Core to each WFU specification is a data model, $\mathcal{D}_{WFU}$, which is updated in the context of a B-Step to evaluate the application logic. The key of $\mathcal{D}_{WFU}$ is a triple (e, t, x), which consists of an external event type e, that occurs at time t for workflow instance x and maintains all attribute values for this B-Step. All other attributes in $\mathcal{D}_{WFU}$ depend on the WFU specification. $\mathcal{D}_{WFU}$ is updated in two ways: by receiving update notifications from other WFUs and by evaluating the application logic $\mathcal{AL}$ over the current state of $\mathcal{D}_{WFU}$ once all relevant attribute updates have been received.

Every single update a WFU is interested in is specified by a subscription $\in \mathcal{S}$ expressed as a set of predicates. A predicate is a triple, (*attributename*, *operator*,

value), to describe an attribute filter, whereby value " $*$ " indicates a wildcard. We also rely on the triple-syntax to express $\mathcal{AL}$ and $\mathcal{N}$. We consider four attribute types: `STATUS` attributes refer to stages and milestones in GSM and are Boolean. `DATA` attributes refer to application data. We assume that data attributes are either text or numeric values. `TOGGLING` attributes represent state transitions of status attributes. The domain is $\{\text{Boolean} \times \text{Boolean}\}$, a pair of Boolean variables. `VISITED` attributes, also Boolean, indicate the success of WFU evaluation in the context of a B-Step. A sample subscription to the toggling of milestone `m` from false to true is:

```
[e,=,*], [t,=,*], [x,=,*], [name,=,m], [type,=,toggling], [value,=,(false,true)]
```

The subscription expresses interest in all events that represent an update to the `toggling` attribute `m` with value `(false,true)`. The predicates for other attributes in the subscription, i.e., `e`, `t`, and `x` are specified as wildcards, denoted by $*$.

5.3.1 Baseline mapping

Next, we provide the BLM mappings for all WFU types. For $\text{WFU}_{\text{SOURCE}}$, WFU_{D}, and WFU_{SINK} we give a concise description, and for WFU_{PAC} and WFU_{S}, we provide more details including a complete example.

$\text{WFU}_{\text{SOURCE}}$ — To register the start of a new B-Step, this WFU subscribes to all external events e from the environment. By means of its application logic, it attaches a timestamp t and locally records (e, t, x) in $\mathcal{D}$. Its notification generates two event types: one event indicates that $\text{WFU}_{\text{SOURCE}}$ has been `visited` for (e, t, x) expressing that the new B-Step has started. In addition, one `data-update` event for each data attribute $d \in \mathcal{I}$ indicates the value of d under e. The latter either contains an updated value, if d was updated with e, or no value else.

WFU_{PAC} — First, all PAC rules, $\mathcal{R}_i \in \mathcal{R}$, with a common prerequisite, π, and consequent, γ, are collapsed into a super PAC rule $(\pi, \mathcal{A}, \gamma)$, where $\mathcal{A} = \bigvee_{\mathcal{R}_i \in \mathcal{R}, \alpha \in R_i} \alpha$ is a disjunction of the individual antecedents. In the following, we use the PAC rule

```
SUBSCRIPTIONS={
  {[e,=,*],[t,=,*],[x,=,*],[name,=,'source'],[type,=,'visited'],[value,=,'true']}
&{[e,=,*],[t,=,*],[x,=,*],[name,=,'bloodValue'],[type,=,'data'],[value,=,*]}
&{[e,=,*],[t,=,*],[x,=,*],[name,=,'S3'],[type,=,'status'],[value,=,*]}
}

APPLICATION LOGIC={
  // prerequisite
  {[id,=,R1],[step,=,'latest'],[name,=,'S3'],[type,=,'status'],[value,=,'true']}
  // antecedent
&{[id,=,R1],[step,=,'current'],[name,=,'e'],[type,=,'data'],[value,=,'T:BloodTest']}
  // antecedent
&{[id,=,R1],[step,=,'current'],[name,=,'bloodValue'],[type,=,'data'],[value,>,42]}
}

NOTIFICATIONS={
  // visited
  {[id,=,P0],[e,=,*],[t,=,*],[x,=,*],[name,=,'WFU+m4'],[type,=,'visited'],[value,=,'true']}
  // successful toggling
||{[id,=,P1],[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)']}
  // no toggling
||{[id,=,P2],[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)']}
}
```

Figure 5.3.1: Example of BLM mapping: $WFU_{\oplus m4}$ for achieving `m4` in patient care business process (cf. Rule 2 in Table 5.1.1).

for achieving milestone `m4` in the patient care process as an example (cf. Rule 2 in Table 5.1.1). The corresponding $\text{WFU}_{\oplus \text{m4}}$ is depicted in Listing 5.3.1. In general, a WFU_{PAC} subscribes to all toggling-, status-, or data attribute updates that occur in π or $\mathcal{A}$ of the super PAC rule (cf. Lines 2-4). The encoding of the PAC rule in the application logic is realized by a conjunction of attribute checks identified by a *ruleID* (R1 in the example). The rule components refer to the current B-Step, i.e., tuple (e, t, x) in $\mathcal{D}$, to evaluate the $\mathcal{A}$ components of the PAC rule (Lines 11 and 13), and to the latest B-Step, i.e., $(e, t-1, x)$ in $\mathcal{D}$, to evaluate the π component (Line 9). The notification contains three possible event types: one indicates that WFU_{PAC} has been `visited` for (e, t, x) (Line 18). The other two templates represent the outcome of rule evaluation — either a `toggling` event for γ if the PAC rule fired due to successful rule evaluation (Line 19), or `no toggling` if the PAC rule did not fire (Line 22).

WFU_S — To manage access to status attributes, this WFU type subscribes to all toggling updates for a particular status attribute s. The application logic dictates when a stable value for s has been reached in B-Step (e, t, x) and when this value can be published to the rest of the system. An example WFU for status attribute $m4$ is depicted in Listing 5.3.2. In general, a stable value for s is either reached if a successful toggling (i.e., `(true,false)` or `(false,true)`) has been received from one of the WFU_{PAC} that control toggling of s (cf. Lines 6 and 8), or if both of these WFU_{PAC}

```
SUBSCRIPTIONS={
   {[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,eq,'toggling'],[value,=,*]}
}

APPLICATION LOGIC={
   {[id,=,R1],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)']}
||{[id,=,R2],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,true)']}
||{[id,=,R3],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,false)']}
||{[id,=,R4],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)']}
}

NOTIFICATIONS={
   // visited
   {[id,=,P0],[e,=,*],[t,=,*],[x,=,*],[name,=,'WFUm4'],[type,=,'visited'],[value,eq,'true']}
   // value of status attribute
||{[id,=,P1],[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,=,'status'],[value,eq,*]}
}
```

Figure 5.3.2: Example of BLM mapping: WFU_{m4} for managing status attribute `m4` in patient care business process.

published that s did not toggle in (e, t, x) (Lines 7 and 9). The notification contains two event templates: one indicates that $\mathrm{WFU_S}$ has been `visited` for (e, t, x), the other indicates the current value of the s as a `status` event. If s was not toggled in (e, t, x), the latest stable value from B-Step $(e, t-1, x)$ is copied to $\mathcal{D}$ and published.

$\mathbf{WFU_D}$ — To manage access to data attributes, this WFU type subscribes to all value updates for a data attribute d. The application logic determines when a stable value for d has been reached in B-Step (e, t, x) and when this value can be indicated to other WFUs. In the BLM mapping, $\mathrm{WFU_{SOURCE}}$ generates a data update event for d under every external event e. If this event contains an updated value, it is persisted in $\mathcal{D}$ for (e, t, x). If no update was attached, the value of d is copied from the latest B-Step. The notification contains two possible event templates: one indicates that $\mathrm{WFU_D}$ has been `visited` for B-Step (e, t, x), the other publishes the stable value of d as a `data` event.

$\mathbf{WFU_{SINK}}$ — To register the completion of a B-Step, this WFU type subscribes to `visited` events from all WFUs generated by the mapping excluding itself: $\mathrm{WFU_{SOURCE}}$, $\mathrm{WFU_{PAC}}$, $\mathrm{WFU_S}$, and $\mathrm{WFU_D}$. Once all notifications are received for (e, t, x), a `visited` event for $\mathrm{WFU_{SINK}}$ is generated to indicate that the B-Step has been completed.

The BLM mapping for the patient care process generates 73 WFUs in total. One $\mathrm{WFU_{SOURCE}}$, 44 $\mathrm{WFU_{PAC}}$, 5 $\mathrm{WFU_D}$, 22 $\mathrm{WFU_S}$, and one WFU_{SINK}.

5.3.2 Context-aware mapping

The rationale of the context-aware mapping (CAM) is to restrict the number of WFUs that must be evaluated within a B-Step to those WFUs that are potentially affected by event e. In BLM, the entire PDG of a GSM model is traversed and every geo-distributed WFU is evaluated in every B-Step. In CAM, we consider the context of e during WFU evaluation. A WFU is only evaluated if:

1. for WFU_{PAC} the PAC rule is reachable over the ePDG for e.
2. for WFU_{D} a data attribute is updated under e.
3. for WFU_{S} a status attribute is updated under e.

Furthermore, for each event type e, CAM generates a single $\text{WFU}^{\text{e}}_{\text{SINK}}$ to check if all WFUs relevant for e have been visited. The advantages of CAM are that fewer WFUs are evaluated, fewer notifications are generated, and the load on WFU_{SINK} is distributed. We now describe the CAM mappings for the five WFU types and point out the differences to the BLM mapping.

$\mathbf{WFU_{SOURCE}}$ — again, this WFU subscribes to all external events, attaches increasing timestamps, and indicates the start of a new B-Step by generating a `visited` event. In contrast to BLM, update notifications to data attributes d are only generated if the event type e is specified in the model to update data attribute d. For all other data attributes, no data update events are generated.

$\mathbf{WFU_{PAC}}$ — Basis is again a super PAC rule. In CAM, we generate the ePDGs for all event types specified in the GSM model to determine those ePDGs (and thereby those event types e) that contain the respective PAC rule in their node set. If the ePDG contains the PAC rule in its node set, it will potentially fire in reaction to e and $\mathcal{AL}$ must be evaluated. Event e is therefore a relevant context for WFU_{PAC}. As in BLM, we refer to the PAC rule for achieving `m4` as an example (cf. Rule 2 in Table 5.1.1). The corresponding CAM $\text{WFU}_{\oplus \text{m4}}$ is depicted in Listing 5.3.3. The only event type that reaches $\oplus$m4 is `T:BloodTest`; therefore, the subscription component

```
SUBSCRIPTIONS={
// ePDG of event contains PAC rule in node set -> event potentially fires PAC rule
    {[e,=,'T:BloodTest'],[t,=,*],[x,=,*],[name,=,'source'],[type,=,'visited'],[value,=,'true']}
// data update relevant for PAC rule evaluation
 || {[e,=,'T:BloodTest'],[t,=,*],[x,=,*],[name,=,'bloodValue'],[type,=,'data'],[value,=,*]}
// ePDG of these events contain PAC rules that update the prerequisite, but firing of this PAC rule
 || {[e,=,'T:BloodTest'],[t,=,*],[x,=,*],[name,=,'S3'],[type,=,'status'],[value,=,*]}
 || {[e,=,'T:Registration'],[t,=,*],[x,=,*],[name,=,'S3'],[type,=,'status'],[value,=,*]}
 || {[e,=,'T:Review'],[t,=,*],[x,=,*],[name,=,'S3'],[type,=,'status'],[value,=,*]}
 || {[e,=,'T:TissueTest'],[t,=,*],[x,=,*],[name,=,'S3'],[type,=,'status'],[value,=,*]}
}

APPLICATION LOGIC={
  // prerequisite update, but no PAC rule firing -> update prerequisite & send visited
  {
   || {[id,=,R0],[name,=,'e'],[type,=,'data'],[value,=,'T:Registration']}
   || {[id,=,R0],[name,=,'e'],[type,=,'data'],[value,=,'T:Review']}
   || {[id,=,R0],[name,=,'e'],[type,=,'data'],[value,=,'T:TissueTest']}
  }
  OR // PAC rule reachable over ePDG -> evalute rule, send result notification & visited
  {
      // prerequisite
      {[id,=,R1],[step,=,'latest'],[name,=,'S3'],[type,=,'status'],[value,=,'true']}
      // antecedent
   &  {[id,=,R1],[step,=,'current'],[name,=,'e'],[type,=,'data'],[value,=,'T:BloodTest']}
      // antecedent
   &  {[id,=,R1],[step,=,'current'],[name,=,'bloodValue'],[type,=,'data'],[value,>,42]}
  }
}

NOTIFICATIONS={ // the same as in the BLM
  // visited
  {[id,=,P0],[e,=,*],[t,=,*],[x,=,*],[name,=,'WFU+m4'],[type,=,'visited'],[value,=,'true']}
  // successful toggling
||{[id,=,P1],[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)']}
  // no toggling
||{[id,=,P2],[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)']}
}
```

Figure 5.3.3: Example of CAM mapping: $WFU_{\oplus m4}$ for achieving `m4` in patient care business process (cf. Rule 2 in Table 5.1.1).

subscribes to all attribute updates necessary for rule evaluation in this context (Lines 3,5,7). In addition, there are also subscriptions to other event contexts since PAC rule evaluation requires the state of the prerequisite attribute, π, from the latest B-Step $(t-1)$. This requires updating π in $\mathcal{D}_{WFU}$ for all event contexts e that reach PAC rules that potentially toggle π. This information can also be extracted from the ePDGs, and WFU subscribes to updates in these event contexts (Lines 7-10). The application logic definition is divided into two cases according to the above contexts: if the context only causes a prerequisite update, i.e., *ruleID* $R0$ (Lines 16-18), $\mathcal{D}_{WFU}$ is updated and a `visited` event is generated (Line 33). If the context triggers the PAC rule (i.e., *ruleID* $> R0$), it is evaluated similar to BLM (Lines 23-27) and an event indicating the result is generated (Lines 35-37). WFU_{PAC} is not involved in B-Steps that are triggered by other event types.

WFU_{S} — In CAM, this WFU type subscribes to toggling updates on status attribute s in the context of such event types e, whose ePDG contains a PAC rule that

potentially toggles s. The subscription is generated similar to $\mathrm{WFU}_{\mathrm{PAC}}$. Compared to BLM, the subtlety in CAM is to determine when exactly the value of s has stabilized under (e, t, x). As a guiding example, we again refer to the WFU for status attribute $m4$, which is depicted in Listing 5.3.4. For some events, both PAC rules are in the ePDG, i.e., $\mathrm{PAC}_{\oplus \mathrm{s}}$ and $\mathrm{PAC}_{\ominus \mathrm{s}}$ (cf. `T:BloodTest` in Lines 20-23). For other events, only one of them is in the ePDG, e.g., `T:Registration` (Lines 10-13). We refer to this as the *toggling cardinality* of s under e. If a real toggling is received for s (e.g., `(true,false)`), the toggling cardinality is irrelevant; s is updated in $\mathcal{D}_{WFU}$ and the stabilized value together with `visited` is indicated. However, if a single no-toggling update was received by a $\mathrm{WFU}_{\mathrm{PAC}}$ (e.g., `(true,true))`, the application logic must know if it can expect another toggling event ($cardinality = 2$), or if the update was the only toggling ($cardinality = 1$). Once the cardinality of updates has been received, the value of s from the latest B-Step is copied and published.

$\mathbf{WFU_D}$ — In contrast to BLM, this WFU type only subscribes to updates on data attribute d in the context of such events e that are specified to update d, and to the start of B-Steps (`visited` of $\mathrm{WFU}_{\mathrm{SOURCE}}$) for such events e that contain PAC rules on their ePDG, which require d as part of the antecedent condition. If a data update is received, the updated value is incorporated into $\mathcal{D}_{WFU}$ and published together with `visited`. If the start of a new B-Step is detected, the value from the latest B-Step is copied and published.

$\mathbf{WFU^e_{SINK}}$ — This WFU type describes the termination of a B-Step for a particular event context e. It subscribes to `visited` events from all WFUs affected by e (cf. CAM mapping for $\mathrm{WFU}_{\mathrm{SOURCE}}$, $\mathrm{WFU}_{\mathrm{PAC}}$, $\mathrm{WFU}_{\mathrm{S}}$, and $\mathrm{WFU}_{\mathrm{D}}$). Once, all notifications are received for (e, t, x), a publication `visited` for $\mathrm{WFU}^{\mathrm{e}}_{\mathrm{SINK}}$ is generated to indicate that B-Step (e, t, x) has been completed.

The CAM mapping for the patient care process generates 82 WFUs in total. One $\mathrm{WFU}_{\mathrm{SOURCE}}$, 44 $\mathrm{WFU}_{\mathrm{PAC}}$, 5 $\mathrm{WFU}_{\mathrm{D}}$, 22 $\mathrm{WFU}_{\mathrm{S}}$, and 10 WFU_{SINK} due to the ten event types specified in the model.

```
SUBSCRIPTIONS={
// ePDG of event contains at least a single PAC rule that potentially fires and updates the status attribute
  {[e,=,'T:Registration'],[t,=,0],[x,=,''],[name,=,'m4'],[type,=,'toggling'],[value,=,*]}
||{[e,=,'T:Review'],[t,=,0],[x,=,''],[name,=,'m4'],[type,=,'toggling'],[value,=,*]}
||{[e,=,'T:BloodTest'],[t,=,0],[x,=,''],[name,=,'m4'],[type,=,'toggling'],[value,=,*]}
||{[e,=,'T:TissueTest'],[t,=,0],[x,=,''],[name,=,'m4'],[type,=,'toggling'],[value,=,*]}
}

APPLICATION LOGIC={{ // differentiate between event types due to toggling cardinality
      {[id,=,'D2'],[e,eq,'T:Registration'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)'],[card,=,1]}
    ||{[id,=,'D3'],[e,=,'T:Registration'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,true)'],[card,=,1]}
    ||{[id,=,'D4'],[e,=,'T:Registration'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,false)'],[card,=,1]}
    ||{[id,=,'D5'],[e,=,'T:Registration'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)'],[card,=,1]}
  } OR {
    ||{[id,=,'D2'],[e,=,'T:Review'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)'],[card,=,1]}
    ||{[id,=,'D3'],[e,=,'T:Review'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,true)'],[card,=,1]}
    ||{[id,=,'D4'],[e,=,'T:Review'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,false)'],[card,=,1]}
    ||{[id,=,'D5'],[e,=,'T:Review'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)'],[card,=,1]}
  } OR {
    ||{[id,=,'D2'],[e,=,'T:BloodTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)'],[card,=,2]}
    ||{[id,=,'D3'],[e,=,'T:BloodTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,true)'],[card,=,2]}
    ||{[id,=,'D4'],[e,=,'T:BloodTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,false)'],[card,=,2]}
    ||{[id,=,'D5'],[e,=,'T:BloodTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)'],[card,=,2]}
  } OR {
    ||{[id,=,'D2'],[e,=,'T:TissueTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,true)'],[card,=,1]}
    ||{[id,=,'D3'],[e,=,'T:TissueTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,true)'],[card,=,1]}
    ||{[id,=,'D4'],[e,=,'T:TissueTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(true,false)'],[card,=,1]}
    ||{[id,=,'D5'],[e,=,'T:TissueTest'],[step,=,'current'],[name,=,'m4'],[type,=,'toggling'],[value,=,'(false,false)'],[card,=,1]}
  }
}

NOTIFICATIONS={
  // visited
  {[id,=,P0],[e,=,*],[t,=,*],[x,=,*],[name,=,'WFUm4'],[type,=,'visited'],[value,=,'true']}
  // value of status attribute
||{[id,=,P1],[e,=,*],[t,=,*],[x,=,*],[name,=,'m4'],[type,=,'status'],[value,=,*]}
}
```

Figure 5.3.4: Example of CAM mapping: WFU_{m4} for managing status attribute `m4` in patient care business process.

5.4 Geo-scale system deployment

The architecture presented in Figure 5.2.1 is divided into two main components: the *management infrastructure* (left) and the *execution infrastructure* (right) [1].

The **management infrastructure** is implemented in the Workflow Manager, which contains a Web Frontend, the Workflow Compiler, and the Workflow Deployment Manager (WDM). The Web Frontend is implemented using the Google Web Toolkit (GWT) and is comprised of a server-side implementation to manage models, including import and export as XML, and a web-based client to graphically represent GSM models. The modeling elements are also used to monitor the execution of individual instances. The Workflow Compiler generates declarative descriptions of WFUs according to the BLM and CAM mappings (cf. Listings 5.3.1, 5.3.2, 5.3.3, 5.3.4), which can also be exported. The deployment of WFUs on geo-distributed computing infrastructures is handled by WDM and organized in two phases: first, WDM forks broker and WF Agent processes on remote machines over SSH and establishes an overlay network. Then, WDM connects to the network and installs WFU descriptions on available WF Agents. To install a WFU at the right geo-destination, every WF Agent subscribes to special control events including its own `agentId`. WDM generates deploy events with the `agentId` of the target and attaches the intended WFU description. WF Agents receive such events, unwrap the WFU, and instantiate it. After WFU deployment, the execution of workflow instances can start.

The **execution infrastructure** of our system is realized in the PADRES geo-scale event management infrastructure and is comprised of a set of WF Agents that are connected to an overlay of pub/sub message brokers [41]. WF Agents are software components that execute the application logic of any WFU and implement the pub/sub interface to communicate with each other over the brokers. A WF Agent parses a WFU description, determines the type of WFU, instantiates the WFU accordingly, and, during execution, handles workflow events as dictated by the WFU.

[1] Demo video available under: `https://www.youtube.com/watch?v=MgZfg8FDJMk`

Algorithm 5.4.1: Process workflow event at WF Agent.

input : wfe = (e, t, x, name, value, type)
Result: process workflow event *wfe* at WFU_{PAC}

1 $r_c \leftarrow \text{LDS}[e, t, x]$ ▷ record for current B-Step
2 $r_l \leftarrow \text{LDS}[e, t-1, x]$ ▷ record for last B-Step
3 $\text{updateRecordWithEvent}(r_c, \text{wfe})$
4 **if** $\text{stabilized}(r_c) \wedge \text{stabilized}(r_l)$ **then**
5 $\quad \mathbb{R} \leftarrow \text{match}(\mathcal{AL}, r_c)$ ▷ match rule logic with B-Step data
6 $\quad \text{updateRecordWithRule}(r_c, \mathbb{R})$ ▷ update with matching result
7 $\quad \text{notify}(r_c, \mathbb{R})$ ▷ publish notifications
8 **return**

The initialization phase is comprised of three steps:

- First, the WF Agent issues all subscriptions from the WFU description to the broker network.
- Next, the WF Agent instantiates a local data store (LDS). The LDS is a multi-versioned representation of the data model $\mathcal{D}_{WFU}$, which is derived from the $\mathcal{S}$ and $\mathcal{N}$ components of the WFU. LDS is implemented as a map that is indexed by the B-Step identifier (e, t, x) and maintains all attributes in $\mathcal{D}_{WFU}$. Every attribute contains three fields: `name`, `type`, and `value`. Toggling attributes also contain a field `count` to store the number of toggling updates received.
- Last, a rule engine to evaluate the application logic of the WFU is instantiated with the rules given in the $\mathcal{AL}$ component of the WFU. The rule engine is based on a rete structure to match facts, i.e., the data collected for B-Steps in LDS, against these rules.

5.4.1 WFU Evaluation at WF Agents

The actual evaluation of a WFU_{PAC} at a WF Agent is described in Algorithm 5.4.1. As input, the WF Agent receives a workflow event `wfe` encoding an attribute update. The event follows the triple-format specification given in Section5.3 and consists

Algorithm 5.4.2: Generate notifications.

input : r, $\mathbb{R}$
Result: publish notifications for record r and set of matching application logic rules $\mathbb{R}$.

```
1 visitedEvent ← instantiate(𝒩[P0], r)
2 publish(visitedEvent)
3 if ℝ ≠ ∅ then
4 |  toggleEvent ← instantiate(𝒩[P1], r)
5 |  publish(toggleEvent)
6 else
7 |  noToggleEvent ← instantiate(𝒩[P2], r)
8 |  publish(noToggleEvent)
9 return
```

of a B-Step identifier (`e`, `t`, `x`), an attribute `name`, `type`, and `value`. First, the key (e,t,x) is extracted from the event, and the corresponding current B-Step record (r_c) is retrieved from LDS (Line 1) and updated with the event content (Line 3). If no record was found, this indicates a new B-Step to the WF Agent and a new record is created; all attribute values are initialized to `null`. The attribute value encoded in `wfe` is then used to update the corresponding attribute in r_c. If the event represents an update of a toggling attribute, the update counter for this attribute is incremented, which is necessary to check the toggling cardinality afterwards. After updating the record, both the current record and the record from the latest B-Step (r_l) are checked for stability (Line 4). A record is stable if all its attributes have been updated for that B-Step and for all toggling attributes the necessary cardinality has been received. If both records are stable, the record r_c is put into the rule engine to determine the set of matching application level rules (Line 5). The matching rules are used to update LDS (Line 6) and to generate events indicating the result to other WFUs (Line 7). Event generation and publication is described in Algorithm 5.4.2. First, a `visited` event is generated (template $P4$ in the $\mathcal{N}$ component of WFUs); if at least one of the rules matches (Line 3), a `toggling` event is generated and published (Line 4 and 5), while if none of them match an event indicating that the attribute will not toggle is generated and published (Line 7 and 8).

Model	Description	#s	PDG	ePDG*	ePDG	BLM*	CAM*
2-Stg-Seq	A sequence of two stages (tasks)	4	8	4	2.6	14	16
4-Stg-Seq	A sequence of four stages (tasks)	8	16	4	3.2	26	30
1-in-3-Split	A parallel split, where termination of one stage opens three other stages	8	16	8	3.2	26	30
1-in-9-Split	A parallel split, where termination of one stage opens nine other stages	20	40	22	3.8	60	72
10-Stg-Seq	A sequence of ten stages (tasks)	20	40	4	3.6	60	72

Table 5.5.1: Synthetic workflow models for evaluation.

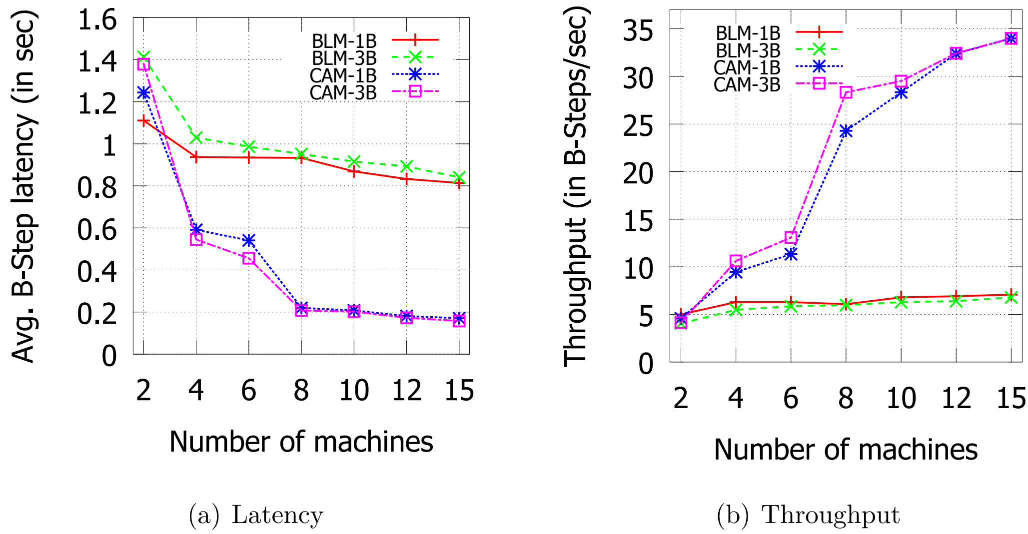
5.5 Experimental Evaluation

We experimentally evaluated our approach in an OpenStack cloud infrastructure. Every virtual machine (VM) was equipped with 2GB RAM, 2 vCPUs @2GHz, and Ubuntu 14.04.1 LTS.

We compared the performance of BLM and CAM for different models, ePDG sizes, broker topologies, and WFU deployments on varying numbers of VMs. In all experiments, we applied two metrics: the *average B-Step latency* refers to the time it takes to compute a single B-Step; the *B-Step throughput* refers to the number of B-Steps executed per unit of time. B-Steps are a good metric because they represent the full impact of an external event e on a GSM instance, i.e., PAC rule firing and attribute mutations. This corresponds to the evaluation of all relevant WFUs in our system including the transmission of notifications. We used two different broker overlay networks: a single broker and a tree structure comprised of three brokers. Each broker was deployed on a dedicated VM. The WF Agents were uniformly distributed over a varying number of VMs and their connections were uniformly distributed across all brokers.

Performance of Patientcare BP — In a first experiment, we compared the performance of our system based on the BLM and CAM mappings of the Patientcare BP. We used both the single broker and the three-brokers-network to connect a varying number of VMs for executing the WFUs. A uniform distribution of external event types was used to trigger 1000 B-Steps. The results are depicted in Figure 5.5.1. In general, CAM outperforms BLM, especially, with increasing numbers of VMs. For instance, for 15 VMs, CAM offers a speed-up of 5X in terms of throughput, and latency is reduced by 80%. This can be explained by the fewer WFUs being involved

and the fewer messages being generated in CAM, as well as by the reduced load on single VMs when scaling out. However, increasing the amount of brokers did not influence performance significantly. Although, the load on a single broker can be reduced as not all publications are routed through all three brokers, however, some publications need two additional hops compared to the single broker case, which negatively effects performance.

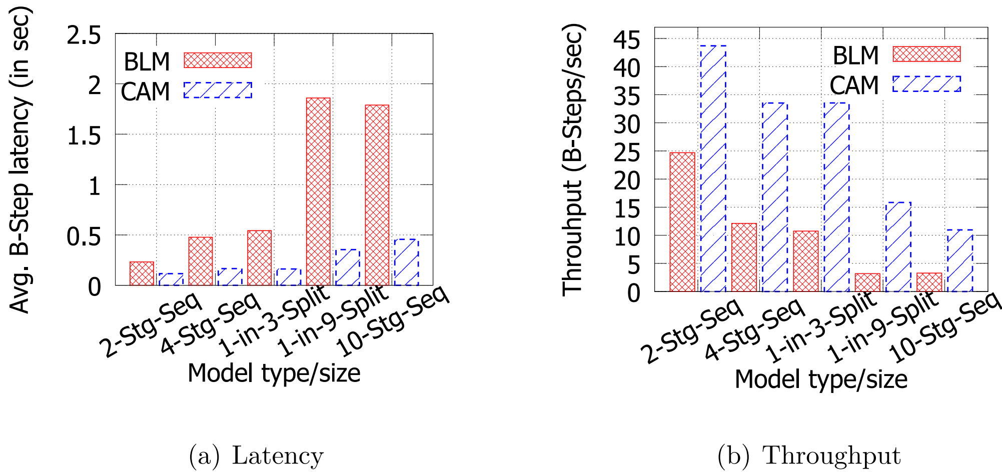


(a) Latency (b) Throughput

Figure 5.5.1: Evaluation of Patientcare business process: impact of mapping type and broker topology on performance.

To obtain insights about factors influencing performance, we also used synthetic models reflecting two core patterns, which are the basis for all workflows: *task sequences* and *parallel task splits/merges*. In GSM, a task sequence is modeled as a sequence of stages, where the achievement of a stage's milestone triggers the next stage. A split is modeled as a stage whose closing opens a set of other stages in parallel. A description of these models is provided in Table 5.5.1. For each model, the number of status attributes (#s), the number of nodes in the PDG (PDG), the longest ePDG (ePDG*), and the average number of nodes in ePDGs (ePDG) is given. In addition, the number of WFUs resulting from both mappings is given (BLM* and CAM*, respectively).

Impact of model type on performance — First, we compared BLM and CAM for all models in Table 5.5.1. For both mappings, the generated WFUs have been uniformly deployed on ten WF Agent VMs and we used a uniform distribution of external events to trigger 1000 B-Steps. The results are depicted in Figure 5.5.2. With increasing model size ($\#s$), both mappings show increased latency and reduced throughput. However, the CAM mapping performs always better, especially, with larger models (about 75% less latency and 400% throughput compared to BLM). This behavior can be explained by the different sizes of PDG (relevant for BLM) and ePDG (relevant for CAM), resulting in less WFUs involved in executing the CAM mapping.

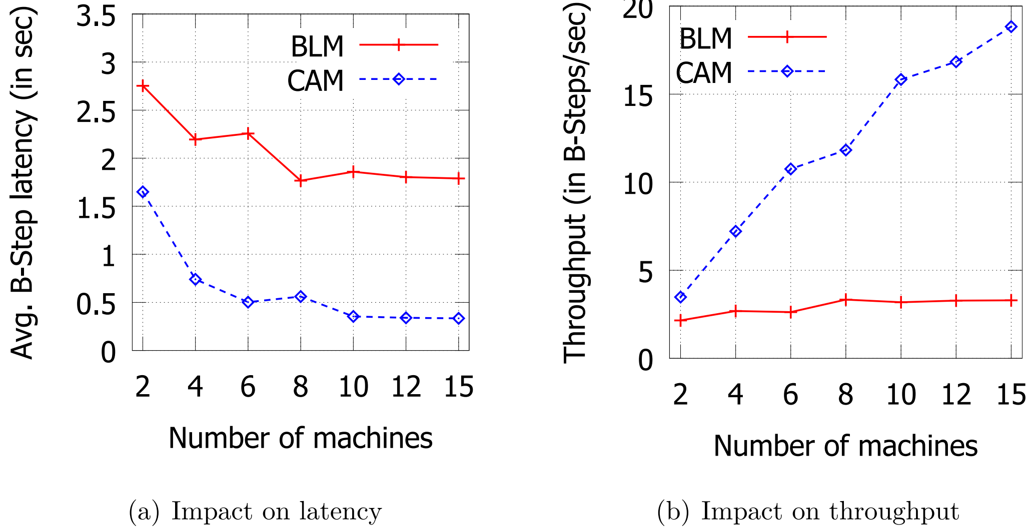


(a) Latency (b) Throughput

Figure 5.5.2: Evaluation of BLM vs. CAM: impact of model type on performance. Fixed deployment on 10 WF agent VMs.

Impact of mapping on scaling — Next, we compared the scaling behavior of BLM and CAM based on model `10-Stg-Seq`. For both mappings, the generated WFUs were uniformly allocated to a varying number of WF Agent VMs. The results are depicted in Figure 5.5.3. With increasing degree of distribution, both mappings offer better performance in terms of latency and throughput. CAM, however, scales much better compared to BLM. For 15 VMs, throughput is 5X the throughput of BLM, while the performance difference for 2 VMs is only 0.5X; latency behavior is similar. This behavior can be explained by the reduced number of WFUs involved in the CAM execution for a particular external event, which provides a better

distribution of the system load for the processing of 1000 B-Steps.

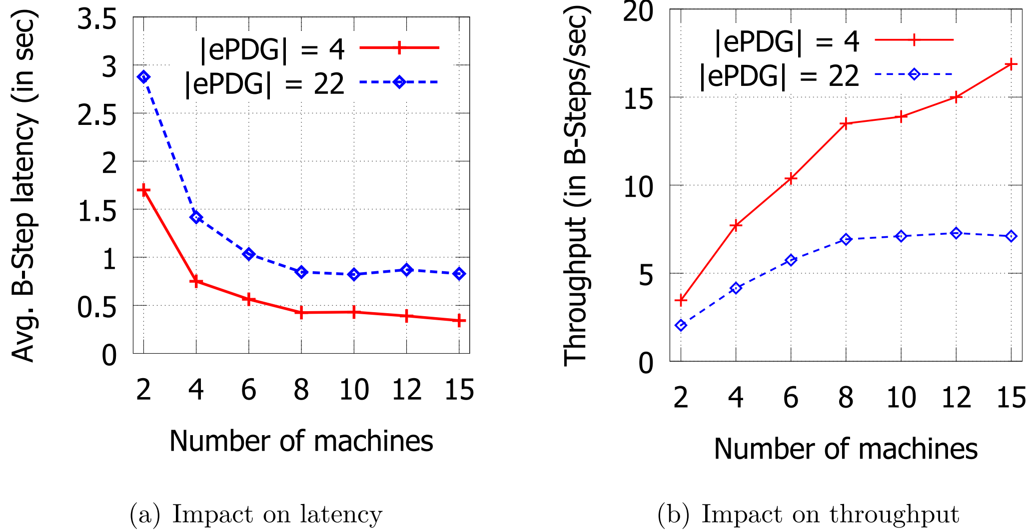


(a) Impact on latency (b) Impact on throughput

Figure 5.5.3: Mapping comparison: impact on scaling performance for model `10-Stg-Seq`.

CAM: impact of ePDG on performance — Last, we focused on the CAM mapping and investigated the impact of the ePDG size on performance. We allocated the WFUs for model `2-Stg-Seq` (ePDG* = 4) and `1-in-9-Split` (ePDG* = 22) uniformly to a varying number of WF Agents. We only used the external event type with the longest ePDG to execute 1000 B-Steps. The results are depicted in Figure 5.5.4 and show that smaller ePDGs have a positive impact on performance, especially, with higher degrees of distribution (throughput tripled for 15 VMs). Again, the explanation for the behavior is the reduced number of WFUs in the B-Step execution, this time, due to the smaller ePDG size.

Evaluation summary — There are three main results obtained from the evaluation: first, both mappings are scalable, i.e., show increased performance w.r.t. B-Step latency and throughput; CAM, however, scales much better than BLM. Second, CAM always offers better performance compared to BLM; especially, sequential task patterns benefit, while parallel splits are costly. Third, in CAM a lower ePDG size for a particular B-Step significantly increases performance.



(a) Impact on latency

(b) Impact on throughput

Figure 5.5.4: Evaluation CAM Mapping: impact of ePDG size.

5.6 Summary

In this chapter, we presented a conceptual architecture and a geo-scale distributed system addressing the combination of two current challenges in business process management: the automated execution of (1) flexible business processes that are (2) geographically scattered across different organizational units. The main problem in such scenarios is to maintain locality of process- and data fragments to provide the privacy of sensitive information, while integrating sub-processes across organizational boundaries.

Our system is founded on the well-established Guard-Stage-Milestone (GSM) meta-model theory, a generalization of flexible business processes and the formal basis of CMMN. We introduced a management infrastructure and a fully distributed execution engine for GSM workflows based on Workflow Units (WFUs), independent system components that represent the unit of distribution in our architecture. WFUs manage individual rules and attributes in GSM workflows and provide a fragmentation of the business process. We presented two mapping implementations of GSM into the WFU representation: The first, our baseline mapping (BLM), directly stems

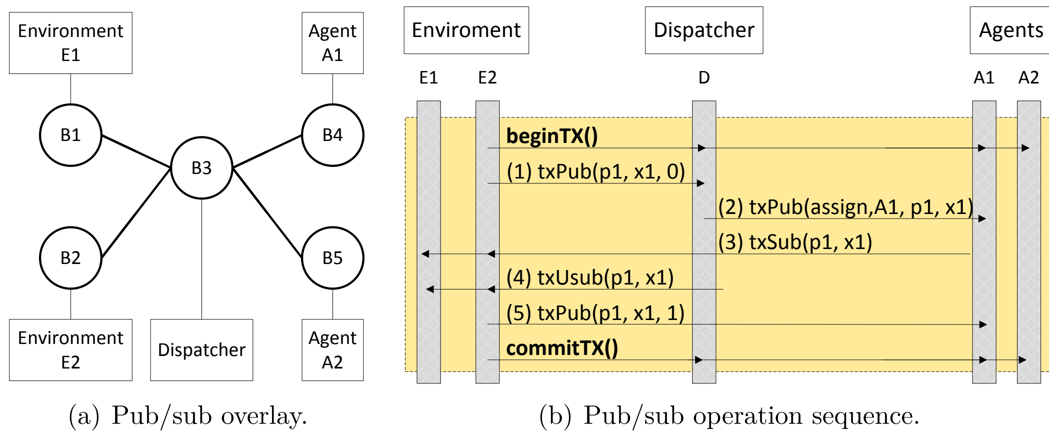
from our theoretical publish/subscribe formulation of GSM presented in Chapter 4. The second, an optimized context-aware mapping (CAM), results from the practical insights we obtained from implementing BLM. In contrast to BLM, CAM considers the type of external events to reduce the computational overhead and network communication delay that are further magnified at geo-scale. Our experimental evaluation in a cloud environment showed that both mappings are scalable but CAM outperforms BLM. In particular, CAM reveals its strengths for sequential task patterns in workflows, while task splits/merges are more costly.

Chapter 6

Multi-client Transactions in Distributed Publish/Subscribe

Publish/subscribe middleware offers a powerful integration and coordination platform for workflow management systems (WFMSs). However, horizontally scaling a WFMS with an increasing number of workflow instances requires a replication of those components in the WFMS that handle individual instances. Replication, however, requires a higher degree of coordination than usually supported by pub/sub.

Use case — In this part of the work, we study the following scenario stemming from an industry-inspired application [49, 90]. It can be realized over a pub/sub architecture as depicted in Figure 6.1(a). A network of five brokers (B1 - B5) connects all clients involved. The WFMS is realized by two workflow *Agents* (A1 and A2). A *Dispatcher* (D) dispatches new instances to one of the agents. The *Environment* clients (E1 and E2) invoke new instances or update existing instances. The sequence diagram in Figure 6.1(b) depicts the dispatching of an instance `x1` for process `p1`, created by E2, through publication `pub(p1,x1,0)`. This publication is routed to D, which, in turn, sends an *assign* publication `pub(assign,A1,p1,x1)` to A1. Now, A1 subscribes to updates for that process instance (`sub(p1,x1)`), while, at the same time, D unsubscribes to such events (`unsub(p1,x1)`). The expected system behavior is that now all update events for `x1` (e.g., `pub(p1,x1,1)`) are only delivered to A1.



(a) Pub/sub overlay.

(b) Pub/sub operation sequence.

Figure 6.0.1: Use case: instance dispatching in WFMS.

However, during the dispatching transition, the destination of update events to `x1` is not guaranteed to be A1. This can happen because the subscription has not been propagated through the whole overlay, i.e., updated all `PRTs` between A1 and E2. Moreover, there is a chance for duplicate delivery of events when the un-subscription by D has not completely been propagated. In summary, the problem is that all four operations need to be executed atomically, in the order imposed by the clients, and isolated from any other concurrent operation, e.g., successive publications to the same instance. These requirements can be captured as ACID semantics for pub/sub.

This chapter presents an approach for realizing multi-client transactions in distributed content-based publish/subscribe systems. Section 6.1 introduces a formalization of the distributed content-based publish/subscribe model, which provides the basis for our formal pub/sub transaction model presented in Section 6.2. Here, we propose different levels of the ACID semantics for expressing multi-user transactions. In Section 6.3, we describe D-TX, our first solution for supporting transactions, which allows a set of operations to be defined at run-time, provides sequential consistency, serializability, and atomicity. S-TX, our second distributed solution, which relies on static knowledge of all operations in a transaction, providing weak isolation, sequential consistency, and atomicity is presented in Section 6.4. Finally, a comprehensive evaluation comparing the strengths of both solutions with a baseline solution, which mimics part of the transaction behavior by manual operation delays, is presented in Section 6.5 before we summarize in Section 6.6.

6.1 Distributed content-based publish/subscribe

The foundation of our approach is the distributed content-based pub/sub model including an *advertisement* optimization for subscription routing [41, 42] that we formalize next.

6.1.1 Event space formalization

The basis of the content-based pub/sub model is a d-dimensional event space $\mathcal{E}_d$, where each dimension is representing an attribute A_i with domain $\mathtt{dom}(A_i)$.

$$\mathcal{E}_d = A_1 \times A_2 \times A_3 \times \cdots \times A_d$$

The domain $\mathtt{dom}(A_i)$ is predefined and ordered; the lowest and highest values are denoted by l_i and u_i, respectively.

A filter F on $\mathcal{E}_d$ is defined as a set of predicates $p_1, \ldots, p_d$, such that the ith predicate represents an interval of values $[p_i^l, p_i^u]$ for attribute A_i within $\mathtt{dom}(A_i)$, where $p_i^l \geq l_i$ and $p_i^u \leq u_i$. With F_p, we refer to a special filter called *point filter*, if and only if, every predicate $p_i \in F_p$ represents a single value from $\mathtt{dom}(A_i)$, i.e., $\forall p_i \in F : p_i^l = p_i^u$. Otherwise, we refer to a filter as a *range filter*, denoted by F_r, if at least a single predicate $p_i \in F_r$ defines an interval with more than a single value, i.e., $\exists p_i \in F : p_i^l < p_i^u$. Two filter relations enable the expression of a matching and a conflict relation among operations: `overlap` (ω: Definition 13) and `subsume` (σ: Definition 14).

Definition 13. Filter overlap: The overlap of filters F_g and F_h is defined as a Boolean function $\omega(F_g, F_h)$ that returns true, if and only if $\exists A_i \in \mathcal{E}_d$ such that $F_g(p_i) \cap F_h(p_i) \neq \emptyset$; i.e., if at least the predicates for a single attribute overlap.

Definition 14. Filter subsumption: The subsumption of a filter F_h by F_g is defined by a Boolean function $\sigma(F_g, F_h)$ that returns true, if and only if $\forall p_i \in F_g, F_h : F_h(p_i^l) \geq F_g(p_i^l) \wedge F_h(p_i^u) \leq F_g(p_i^u)$; i.e., each predicate $\in F_g$ is an interval containing the predicate in the other filter (F_h).

6.1.2 Elementary publish/subscribe operations

Based on the event space formalization $\mathcal{E}_d$ and the filter concept F, we now formalize the set of elementary pub/sub operations. We assume that an operation is issued by a client $c_i \in C = \{c_1, \ldots, c_n\}$.

$$\mathcal{O}_{PS} = \{\texttt{pub}(F_p), \texttt{adv}(F), \texttt{uadv}(F), \texttt{sub}(F), \texttt{usub}(F)\}$$

where F and F_d represent filters on $\mathcal{E}_d$. To refer to a particular client, c_i, issuing operation $\texttt{op} \in \mathcal{O}_{PS}$, we write $\texttt{op}[c_i](F)$; operations have the following semantics:

- $\texttt{pub}(F_p)$ – publishes an event that is represented by a point filter F_p on $\mathcal{E}_d$, i.e., it describes a single value from the domain of each attribute $A_i \in \mathcal{E}_d$.
- $\texttt{adv}(F)$ – advertises events a client will publish in the future. The set of advertisements, $\mathcal{A}_c$, a client issued forms the clients publication space: $F^c_{pub} = \bigcup_{F_i \in \mathcal{A}} F_i$. Every publication, $\texttt{pub}[c](F_p)$, must be matched by F^c_{pub}, such that $\sigma(F^c_{pub}, F_p)$ returns true.
- $\texttt{uadv}(F)$ – unadvertises a prior advertisement $\texttt{adv}(F)$ of a client. Effectively, an unadvertisement reduces the client publication space to $F^c_{pub} = \bigcup_{F_i \in \mathcal{A}} F_i \setminus F$.
- $\texttt{sub}(F)$ – subscribes to publications from other clients, $\texttt{pub}(F_p)$, that match the subscription, i.e., $\sigma(F, F_p)$ returns true. The set of subscriptions, $\mathcal{S}_c$, a client issued forms the clients subscription space: $F^c_{sub} = \bigcup_{F_i \in \mathcal{S}} F_i$.
- $\texttt{usub}(F)$ – unsubscribes a prior subscription $\texttt{sub}(F)$ of a client. Effectively, an unsubscription reduces the client subscription space to $F^c_{sub} = \bigcup_{F_i \in \mathcal{S}} F_i \setminus F$.

A notification represents the delivery of a publication, $\texttt{pub}(F_p)$, to an interested client c if $\sigma(F^c_{sub}, F_p)$ returns true. The set of notifications a client received over time is represented by $\mathcal{N}_c$.

6.1.3 Broker network

A network of brokers, $B = \{b_1, \ldots, b_n\}$ is used to store, process, and forward pub/sub operations. The processing mainly involves matching different operations and can be expressed by the overlap function $\omega(F_g, F_h)$ and the subsumption function $\sigma(F_g, F_h)$. Advertisements, $\texttt{adv}(F)$, are broadcast to all brokers and stored in a Subscription Routing Table (SRT), i.e., a list of $[\texttt{adv}, \texttt{lastHop}]$-pairs, where `lastHop` points to a broker or client that sent the operation. A subscription $\texttt{sub}(F_s)$ is matched against all $\texttt{adv}(F_a) \in \texttt{SRT}$ at a broker, where a successful match is defined by $\omega(F_s, F_a) = \top$. Matching subscriptions are stored in a Publication Routing Table (PRT), which is a list of $[\texttt{sub}, \texttt{lastHop}]$-pairs, and forwarded to the `lastHops` of the matching advertisements. Non-matching subscriptions are buffered until they match a later advertisement and get forwarded (i.e., the advertisement attracted the subscription); a publication $\texttt{pub}(\texttt{F}_\texttt{p})$ is matched against all $\texttt{sub}(F_s) \in \texttt{PRT}$, where a successful match is defined by $\sigma(F_s, F_p) = \top$. Not matching publications are dropped; matching publications are forwarded to the `lastHops` of the matching subscriptions and eventually clients are notified.

6.2 Transactional content-based publish/subscribe model

We now present our formal transaction model for distributed content-based pub/sub systems (cf. Section 6.1). We first present our definition of transactions in the context of pub/sub and the ACID properties. We then focus on two specific properties, consistency and isolation, and demonstrate how they can be supported in a distributed pub/sub system with multi-user transactions.

6.2.1 Definition and properties of a transaction

A pub/sub transaction, $\mathcal{T}_{PS}$, consists of a sequence of elementary pub/sub operations $o_1, \ldots, o_n$, where $o_i \in \mathcal{O}_{PS}$. Each operation is a pub/sub operation, which originates from any client in the system. In this way, a transaction can involve *multiple users*, with operations originating from different sources. Additionally, each transaction is *distributed*, since the pub/sub operations must be applied at various brokers, which communicate only using overlay links.

In Definition 15, we define the ACID semantics for a pub/sub transaction $\mathcal{T}_{PS} = \{o_1, \ldots, o_n\}$, $o_i \in \mathcal{O}_{PS}$, executed on a pub/sub system $\Pi = (\mathbb{B}, \mathbb{C})$, where a set of clients $\mathbb{C} = \{c_1, \ldots, c_m\}$ is connected to a broker network $\mathbb{B} = \{b_1, \ldots, b_k\}$.

Definition 15. ACID semantics in pub/sub. We define the ACID semantics for a transaction $\mathcal{T}_{PS}$ as follows:

- *atomicity* – either all operations $\in \mathcal{T}_{PS}$ are successfully processed by Π or none of them. In particular, the `SRT`s of brokers are updated with the `adv/uadv` operations, the `PRT`s are updated with the `sub/usub` operations, and clients are notified with publications `pub` $\in \mathcal{T}_{PS}$.
- *consistency* – $\mathcal{T}_{PS}$ transitions a correct pub/sub system state Σ into another correct state Σ^* (cf. Definition 16). A correct state transition by $\mathcal{T}_{PS}$ is defined through *internal* and *external* consistency:
  1. *internal consistency:* every operation `op` $\in \mathcal{T}_{PS}$ is executed on a consistent state Σ of Π. In particular, the states of `SRTs` and `PRTs` are consistent across all brokers $b \in \mathbb{B}$ while processing `op`[1].
  2. *external consistency:* the order in which the operations of $\mathcal{T}_{PS}$ are processed by Π is prescribed by the order expressed by the application semantics.

[1] To better understand internal consistency consider an overlay with two brokers b_1 and b_2. A publication, p_1, first arrives at b_1, where it is processed based on the `PRT` of b_1, and is then forwarded to b_2. If the `PRT` of b_2 represents a different state compared to the `PRT` of b_1 consistency is violated. A different state might be reached because a subscription was concurrently processed by b_2 and modified its `PRT`.

- *isolation* – $\mathcal{T}_{PS}$ only reads the latest committed state. In particular, any `sub/usub` only reads the `SRTs` and any `pub` only reads the `PRTs` of brokers constructed by the latest committed transaction (*serializability*).
- *durability* – a committed transaction, i.e., all routing state changes in $\mathbb{B}$ and all event notifications in $\mathbb{C}$ are durable and survive node and network failures.

Definition 16. Consistent pub/sub system state: A consistent pub/sub system state Σ for $\Pi = (\mathbb{B}, \mathbb{C})$ is defined as the state that is reached at all brokers $b_i \in B$, i.e., `SRTs` and `PRTs`, and all clients $c_i \in \mathbb{C}$, i.e., F^c_{pub}, F^c_{sub}, and $\mathcal{N}_c$, after completely applying a single operation $\texttt{op} \in \mathcal{O}_{PS}$.

Atomicity is taken into consideration in our designs (see Section 6.3 and 6.4). For durability, we employ existing techniques for tolerating broker failures such as [86], which will not be described in this work. For consistency, the main challenge is maintaining external consistency for multi-user transactions, since pub/sub clients are acting in an asynchronous and decoupled manner. For isolation, the main challenge comes from the distributed nature of the pub/sub system, which can be modeled as a replicated database, where each broker maintains a partial copy of the pub/sub state.

6.2.2 Multi-user consistency

An inherent prerequisite to reason about consistency in multi-user transactions is a definition of a working order between operations from different clients. Therefore, we first introduce a special operation $\texttt{tcm}(F_p)$ (transaction control message) and add it to the set of elementary pub/sub operations, $\texttt{tcm}(F_p) \in \mathcal{O}_{PS}$. A `tcm` is a special type of publication specified to trigger other operations at the receiving client, e.g., a client receiving a `tcm` issues a subscription `sub`. Our model provides three different mechanisms to express an order between operations, as specified in Definition 17.

Definition 17. Transaction construction: A transaction $\mathcal{T}_{PS}$ has a coordinator TXC $\in \mathbb{C}$ and identifier `txID`. The TXC issues the first operation together with `txID`; then, the complete operation sequence is constructed by:

1. *TXC operation* – the TXC issues more operations $\in \mathcal{O}_{PS}$ to B using `txID`.
2. *TCM-triggered operation* – a client that received a `tcm` for `txID`, issues its own operations using `txID`.
3. *Internally-triggered operation* – a broker that received an `adv` (or `uadv`) operation matching a buffered subscription forwards the `sub` (or `usub`) using `txID`.

Consistency – In our model, a consistent state transfer is defined by *internal* and *external* consistency. Our formulation of the pub/sub ACID semantics defines a consistent state transfer through *internal* and *external* consistency. While internal consistency is already precisely described, external consistency requires a consistency model. Essentially, all operations should be executed according to an application-specific order, i.e., the sequential order, in which clients issued them (cf. Definition 18).

Definition 18. Sequential consistency: External consistency in $\mathcal{T}_{PS}$ is defined as a sequential order relation among operations, denoted $<^{SO}$, by the following rules:

1. *Thread*—for any two operations $\mathtt{a}, \mathtt{b} \in \mathcal{T}_{PS}$ issued by the same client: if `a` happened before `b`, then $\mathtt{a} <^{SO} \mathtt{b}$.
2. *Trigger*—for two operations $\mathtt{a}, \mathtt{b} \in \mathcal{T}_{PS}$: if `a` triggers `b`, where `b` is either an *internally-triggered-operation* at a broker or a *TCM-triggered-operation* at a different client, then $\mathtt{a} <^{SO} \mathtt{b}$.
3. *Trigger** – for three operations `a`, `b`, and `c`: if $\mathtt{a} <^{SO} \mathtt{b}$ according to rule *Trigger*, and if $\mathtt{a} <^{SO} \mathtt{c}$ according to rule *Thread*, then $\mathtt{b} <^{SO} \mathtt{c}$. In other words, every triggered operation is ordered prior to any subsequent operation from the same client.
4. *Transitivity* – for any three pub/sub operations `a`, `b`, and `c`, if $\mathtt{a} <^{SO} \mathtt{b}$ and $\mathtt{b} <^{SO} \mathtt{c}$, then $\mathtt{a} <^{SO} \mathtt{c}$.

For two operations $\mathtt{a} <^{SO} \mathtt{b} \in \mathcal{T}_{PS}$, the pub/sub system $\Gamma = (\mathbb{B}, \mathbb{C})$ must completely process `a` before `b`. In particular, every broker $b \in \mathbb{B}$ must process `a` before `b`.

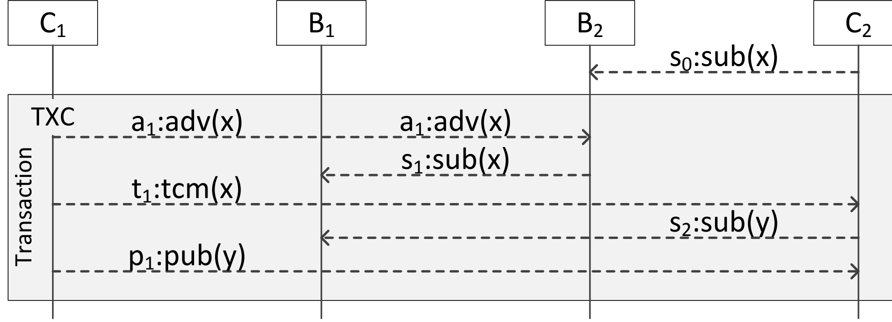


Figure 6.2.1: Example for sequential consistency.

An example for consistent ordering of operations in a transaction is depicted in Figure 6.2.1. In this example, a subscription s_1 by client C_2 is buffered at B_2 before the actual transaction starts. The subscription is buffered due to a lack of matching advertisements at B_2. In the transaction, client C_1 acts as TXC and issues three operations: first, advertisement $a_1 = \texttt{adv}(\texttt{x})$, followed by TCM $t_1 = \texttt{tcm}(\texttt{x})$, and publication $\texttt{pub}(\texttt{y})$. According to rule *Thread*, the TXC operations are ordered $a_1 <^{SO} t_1 <^{SO} p_1$. According to rule *Trigger*, $a_1 <^{SO} s_1$, i.e., the advertisement a_1 attracts the subscription s_1, and $t_1 <^{SO} s_2$, i.e., the TCM triggers subscription s_2; and according to rule *Trigger**, $s_1 <^{SO} t_1$ and $s_2 <^{SO} p_1$, which results in the following processing sequence of operations: a_1, s_1, t_1, s_2, p_1.

6.2.3 Distributed isolation

We define isolation in our pub/sub model by specifying, for two concurrent transactions, which updates made by one transaction should be visible to the other transaction.

In pub/sub, updates refer to changes in the routing state of brokers, i.e., `SRT` and `PRT`. (Un-) advertisements update the `SRT`: every `adv`() writes an entry and every `uadv`() removes an entry from the `SRT`. (Un-) subscriptions read from the `SRT` and update the `PRT`: every `sub`() writes an entry and every `usub`() removes an entry from the `PRT`. Similarly, publications `pub` read the `PRT`.

A major problem of not properly isolating transactions in distributed pub/sub is that publications might get delivered to unintended recipients. Consider, for example, a scenario in which two transactions $t_1 = \{\texttt{sub}[c_2](x), \texttt{usub}[c_3](x)\}$ and $t_2 = \{\texttt{pub}[\texttt{c}_1](\texttt{x})\}$ concurrently execute. Transaction t_1 can be seen as the intent to move a subscription $\texttt{sub}(x)$ from client c_3 to client c_2, so that any of the following publications is no longer notified to c_3 but to c_2. In general, there are three ways to schedule the operations of t_1 and t_2:

1. $\texttt{pub}[c_1](x), \texttt{sub}[c_2](x), \texttt{usub}[c_3](x)$
2. $\texttt{sub}[c_2](x), \texttt{pub}[c_1](x), \texttt{usub}[c_3](x)$
3. $\texttt{sub}[c_2](x), \texttt{usub}[c_3](x), \texttt{pub}[c_1](x)$

Only the first and the third schedule describe the intended system behavior. In the second schedule $\texttt{pub}[c_1](x)$ is routed to both c_2 and c_3 because (1) $\sigma(\texttt{sub}[c_2](x), \texttt{pub}[c_1](x))$ and $\sigma(\texttt{usub}[c_3](x), \texttt{pub}[c_1](x))$, i.e., the operations are conflicting, and (2) the unsubscription $\texttt{usub}[c_3](x)$ was not processed yet. In fact, the PRT state which was read for x in order to process $\texttt{pub}[c_1](x)$ was consistent but it was not committed and subject to further changes by t_1.

To avoid this, every transaction must always read the latest committed state from SRTs and PRTs, defined as *serializability.* For two transactions t_1 and t_2, if t_1 commits before t_2 (denoted $t_1 < t_2$), then all operations $\in t_2$ must be processed based on state written by t_1; in particular, if t_1 changed the routing state for a filter F, then t_2 must read the updated state.

6.3 Dynamic Transaction Service

In this section, we describe the design and implementation of our D-TX approach. D-TX supports tree-based broker overlays and dynamic pub/sub transactions, where a TXC initiates the transaction with an arbitrary operation (e.g., a tcm), unaware

of operations by other clients (e.g., a `sub` triggered by the `tcm`). Atomicity is achieved by adapting the 2-phase commit protocol to distributed pub/sub systems. Isolation is realized through a snapshot isolation algorithm and optimistic concurrency control. For sequential consistency, we propose an acknowledgment-based approach to guarantee the working order of pub/sub operations.

Frequently used acronyms, components, and message formats are summarized and briefly explained in Table 6.3.1.

Table 6.3.1: D-TX notation.

`txID`	transaction identifier, tuple (`clientID` + `txSeqNo`)
`RS`	stable routing state of broker, i.e., `SRT` and `PRT`
`SSID`	version of stable routing state `RS` at broker
`TXContext`	context for particular transaction, based on snapshot of latest committed transaction (`baseSSID`)
`TXRS`	isolated routing state for a particular transaction
`txMap`	index structure to map `txIDs` to `TXContexts`
`txDependencies`	structure to maintain transaction commit order
`ackMap`	structure to manage pending acknowledgments
`TXManager`	transaction handler component of client
`tcm(`F_p`)`	TCM operation (using point filter)
`initMsg(txID,ssID)`	initialization message for `txID`
`initAckMsg(txID,ssID)`	acknowledgment for `initMsg()`
`opMsg(txID,opID,op)`	operation message for `op` $\in \mathcal{O}_{PS}$
`ackMsg(txID,opID)`	acknowledgment for `opMsg()`
`prepMsg(txID)`	prepare message for transaction
`prepAckMsg(txID,status)`	acknowledgment for `prepMsg()`
`cmtMsg(txID)`	commit message for transaction
`abtMsg(txID)`	abort message for transaction

6.3.1 D-TX overview

Processing a transaction in D-TX is staged into three phases and invoked by the TXC. The purpose of the *initialization phase* is two-fold: first, all brokers in the overlay are informed about the new transaction and agree on a common snapshot (i.e., the latest committed transaction) to create the transaction context. Second, a commit

order among concurrent transactions is established. After initializing the transaction, the actual pub/sub operations are processed in the *operation phase*. Sequential operation processing is achieved through a nested acknowledgment mechanism that notifies the TXC once all operations have successfully been applied. The final *termination phase* adopts a 2-PC protocol, tailored to distributed pub/sub systems, to commit the transaction: first, in a *prepare* round, all brokers are informed about the commit intent. Then, every broker verifies if all transactions ordered prior to the committing one have terminated and whether conflicts exist or not. If conflicts exist, the transaction aborts, otherwise, the transaction commits (*commit* round).

6.3.2 D-TX client design

Now, we present the client design of D-TX: We describe the client interface and the concepts relevant to managing transactions. Every client can take one of two roles in the context of a transaction: as the coordinator of the transaction (TXC), or as a participant in the transaction (TXP).

Client API – D-TX extends the standard pub/sub interface with a callback handler *notify()* to deliver incoming publications to an application (cf. Table 6.3.2).

Table 6.3.2: Client interface.

`beginTX():String`	initialize transaction, returns `txID`
`commitTX(txID):status`	terminate tx: returns `comitted` or `aborted`
`txTCM(txID,`F_p`):void`	send `tcm` using `txID`
`txPub(txID,`F_p`):void`	publish using `txID`
`txAdv(txID,`F_r`):void`	advertise using `txID`
`txUadv(txID,`F_r`):void`	un-advertise using `txID`
`txSub(txID,`F_r`):void`	subscribe using `txID`
`txUsub(txID,`F_r`):void`	un-subscribe using `txID`
`notify():publication`	callback: returns publication received

In addition, the method *txTCM()* enables the construction of distributed transactions. Like a publication, *txTCM()* specifies a single point from an event space, while the remaining operations take a range filter as input. Every operation is labeled with

the `txID` of the associated transaction. *beginTX()* triggers the *initialization phase* through an `initMsg` and returns a `txID`; *commitTX(txID)* terminates the transaction.

Our use case scenario can be implemented as sketched in Figure 6.1(b), where operations (1) and (2) are realized with *txTCM()*.

Transaction management – Internally, a client maintains a transaction manager, `TXManager`, for every transaction it is involved in. `TXManager` invokes the initialization and termination protocol and manages the sequential execution of operations. To this end, it maintains a data structure `operations`, which stores all operations received from client API calls. Every operation is described by a tuple (`seqNo`, `opID`, `op`, `status`), where `seqNo` is a sequence number for local operations, `opID` is a globally unique operation identifier, `op` is the actual operation, and `status` describes whether the operation has already been processed (acknowledged), is currently being processed (active), or is buffered due to pending prior operations.

Before any operation is processed, `TXManager` must complete the *initialization phase*, i.e., wait for the corresponding acknowledgment (`initAckMsg`). Similarly, in order to terminate a transaction, the `TXManager` waits for all operations being processed before invoking the *termination phase* through a `prepMsg`. Depending on the outcome, it returns `committed` or `aborted`.

6.3.3 D-TX broker design & protocol

An overview of the broker architecture is depicted in Figure 6.3.1. A broker has a single input queue for incoming messages and a dedicated output queue for each connected client or broker. The main component of a broker is the transactional router (`TXRouter`) handling protocol messages for all three phases in FIFO order within a single thread.

`TXRouter` maintains a stable version of the routing state (`RS`), which reflects the `SRT` and `PRT` generated by the latest committed transaction; `SSID` refers to the version, i.e., the `txID` of the transaction that produced `RS`. On initialization, an isolated base

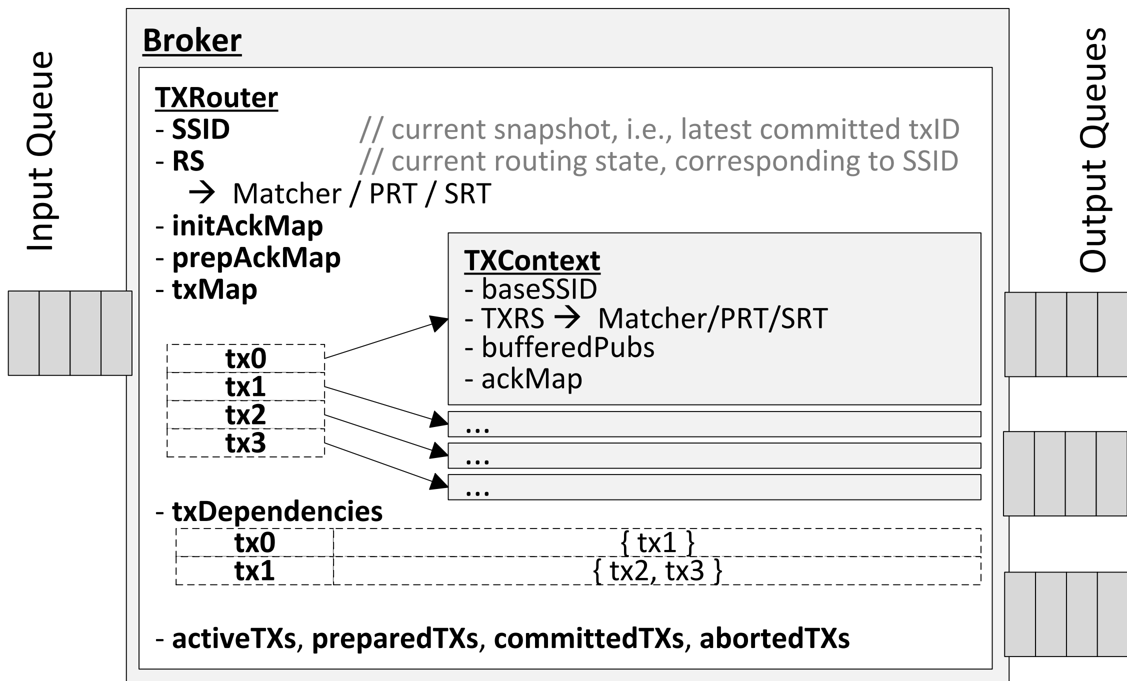


Figure 6.3.1: Internal broker architecture in D-TX.

snapshot, i.e., a copy of `RS`, is taken to form a new transaction context (`TXContext`). This context is used to process all operations of the transaction using its transactional routing state `TXRS`; every operation is either processed by `TXRS` or updates `TXRS`. On commit, `TXRS` is checked for conflicts with concurrent transactions ordered prior to the current one; if no conflicts are detected, it is merged with `RS`, else, it is discarded and the transaction aborts.

To keep track of all transactions and their dependencies, `txMap` provides access to each `TXContext` and `txDependencies` captures the commit order among concurrent transactions. For each `txID`, it stores references to all transactions whose base snapshot is taken from `txID`. The order among concurrent transactions in this list is implicitly given through the lexicographical order of their `txIDs`. In Figure 6.3.1, for instance, a transaction $tx1$ serves as a base snapshot for $tx2$ and $tx3$, where $tx2 < tx3$. A transaction can be in various states: `activeTXs` indexes all active transactions, i.e., transactions that still process operations, `preparedTXs` indexes all transactions that are currently in the *prepare* phase, and `committedTXs` and `abortedTXs` maintain a history of committed and aborted transactions, respectively.

Initialization phase – The purpose of this phase is to inform all brokers about a new transaction, agree on a common base snapshot and establish a consistent order among concurrent transactions. The initialization protocol is shown in Algorithm 6.3.1.

A broker receiving an initialization message (`initMsg`) from the TXC, determines the base snapshot version for the transaction (`ssID`) based on its stable state (`SSID`) and forwards `initMsg(ssID)`. Every broker without further neighboring brokers generates a new transaction context, adds the transaction to the set of active transactions, and responds with an acknowledgment `initAckMsg(txID,ssID)`. To keep track of whether all neighboring brokers have initialized the transaction properly, every broker maintains an `initAckMap` with entries for every broker the `initMsg` has been forwarded to. After receiving all pending acknowledgments, a broker initializes the transaction itself and sends an acknowledgment to the origin of `initMsg`. Due to the tree structure of the broker overlay, message propagation is acyclic and the TXC eventually receives an `initAckMsg` indicating that all brokers have initialized the transaction. In general, it is possible that during the initialization phase other transactions concurrently commit. However, the protocol is still safe because a base snapshot was selected that has been committed at every broker in the network.

Operation phase – The purpose of this phase is to process all operations of a transaction constructed according to Definition 17, i.e., in the order specified by participating clients. The protocol is shown in Algorithm 6.3.2.

Handle operation message – The `TXManager` of a client generates an operation message, `opMsg = (txID,opID,op)` and sends it to the broker it is connected to. Brokers receiving an `opMsg` from some broker/client `origin`, access the corresponding `TXContext` and generate an entry, `oEntry`, for `op` in the `ackMap`. `oEntry` keeps track of whether the broker can acknowledge processing of the operation to `origin`.

First, every broker checks if `op` triggers further internal operations, `tOp`, such as buffered subscriptions (Line 6). If yes, another entry, `tEntry`, with a dependency to `op` is created, and `tOp` is added as pending operation to `oEntry` (Lines 8–11). The broker must then wait for the completion, i.e., acknowledgment, of `tOp` before acknowledging `op` itself. The triggered operation is now forwarded to the newly matching destinations and pending ack references are added to `tEntry` (Lines 12–14).

Algorithm 6.3.1: Initialization phase in D-TX.

```
1  Procedure handleInitMsg(txID, ssID) from origin
2      if origin = CLIENT then  ssID = SSID
3      iEntry ← createInitAckEntry(ssID, origin)
4      initAckMap.put(txID, iEntry)
5      neighbors = brokerNeighbors \ origin
6      if neighbors = ∅ then
7          txCtx ← newTXContext(txID, ssID)
8          txMap.put(txID, txCtx)
9          activeTXs.add(txID)
10         ackMsg ← createInitAckMsg(txID, ssID)
11         send ackMsg to origin
12     else
13         forall broker ∈ neighbors do
14             iEntry.addPendingAck(broker)
15             forward txInitMsg to broker
17
18 Procedure handleInitAckMsg(txID, ssID) from origin
19     iEntry ← initAckMap.get(txID)
20     iEntry.removePendingAck(origin)
21     if iEntry.getPendingAcks() = ∅ then
22         txCtx ← newTXContext(txID, ssID)
23         txMap.put(txID, txCtx)
24         activeTXs.add(txID)
25         forward initAckMsg to iEntry.getOrigin()
26     else
                                   // wait until all ACKs are receved.
```

Algorithm 6.3.2: Pub/Sub operation processing in D-TX.

```
1  Procedure handleOpMsg(txID, opID, op) from origin
2      ctx ← txMap.get(txID)
3      ctx.add(opID, op)                                   // add op to TXRS
4      oEntry ← createOpAckEntry(opID, origin)
5      ctx.getAckMap.put(opID, oEntry)
6      triggeredOps ← ctx.getMatchingBufferedOps(op)
7      forall tOp ∈ triggeredOps do
8          tEntry ← createOpAckEntry(tOp.opID, tOp.origin)
9          ctx.getAckMap.put(tOp.opID, tEntry)
10         oEntry.addPendingOp(tOp.opID)
11         tEntry.addDependency(opID)
12         forall tDest ∈ tOp.getDestinations() do
13             tEntry.addPendingAck(tDest)
14             forward tOp to tDest
15     oDestinations ← ctx.getRoutingMatches(op)
16     forall oDest ∈ oDestinations() do
17         oEntry.addPendingAck(oDest)
18         forward op to oDest
19     if oDestinations ≠ ∅ ∧ triggeredOps ≠ ∅ then
20         send ackMsg(txID, opID) to origin
22
23 Procedure handleOpAckMsg(txID, opID) from origin
24     ctx ← txMap.get(txID)
25     oEntry ← ctx.getAckMap.get(opID)
26     oEntry.removePendingAck(origin)
27     if oEntry.getPendingAck = ∅ then
28         send ackMsg(txID, opID) to oEntry.getOrigin()
29         forall dOpID ∈ oEntry.getDependencies() do
30             dEntry ← ctx.getAckMap.get(dopID)
31             dEntry.removePendingOp(opID)
                       // check if dOp is acknowledged, forward ack.
```

Second, the forward destinations for the actual operations are determined by matching `op` with the transaction's routing state `TXRS`. Then, `op` is forwarded and corresponding pending ack references are added to `oEntry` (Lines 15–18). In addition to forwarding, publications are also buffered in the `TXContext` of every broker on their path to subscribers. This is necessary to determine potential conflicts to prior-ordered transactions later on.

If `op` does not trigger any internal operation, nor is forwarded `op` to anyone else (e.g., at an edge broker for a subscription), an acknowledgment message, `opAckMsg = (txID, opID)`, is generated and returned to `origin`.

Handle acknowledgment message — A broker that receives an `ackMsg` from some client/broker `origin` removes the corresponding entry from its `ackMap`. If all acknowledgments have been received, `ackMsg` is forwarded to the origin of `op` (Lines 24–28). For all operations, `dOp`, that depend on `op`, the corresponding reference is removed from their `ackMap` entry. If `dOp` has no more pending acknowledgments or operations, it can also forward an `ackMsg` (Lines 29–31).

The same mechanism for sequential operation processing is applied at clients. If a client, for instance, receives a `tcm` and issues a `sub` in turn, it waits for an `ackMsg` for the subscription before acknowledging the `tcm`. For a TXC, the operation phase is completed once it receives the acknowledgment for its last operation.

Termination phase – Transaction termination is realized in two rounds: in the *prepare* round, potential conflicts are identified. Based on the outcome, in the *commit* round, the transaction either commits, or aborts. The algorithm for the *prepare* round is shown in Algorithm 6.3.3.

The TXC generates a prepare message, `prepMsg(txID)`, and sends it to its edge broker. Similar to the *initialization*, the `prepMsg` is flooded through the overlay and opposite edge brokers respond with acknowledgments, which are collected at intermediary brokers to eventually notify the TXC about the completion of this round and its outcome.

First, a broker receiving a `prepMsg` verifies if all prior-ordered concurrent transactions

Algorithm 6.3.3: Prepare phase in D-TX.

```
1  Procedure handleTXPrepareMsg(txID) from origin
2  |  if not all prior-ordered transactions terminated then
3  |  |  preparedTX.put(txID, origin)                 // buffer and wait.
4  |  else
5  |  |  handlePreparedTX(txID, origin)
7
8  Procedure handlePreparedTX(txID, origin)
9  |  pEntry ← createPrepAckEntry(txID, origin)
10 |  prepAckMap.put(txID, pEntry)
11 |  if detectConflictsWithPriorTXns (bufferedPubs) ≠ ∅ then
12 |  |  pEntry.setStatus(FALSE)
13 |  else
14 |  |  pEntry.setStatus(TRUE)
15 |  neighbors ← brokerNeighbors \ ori
16 |  if neighbors = ∅ then
17 |  |  ackMsg ← newPrepAckMsg(txID, pEntry.getStatus())
18 |  |  send ackMsg to origin
19 |  else
20 |  |  forall broker ∈ neighbors do
21 |  |  |  pEntry.addPendingAck(broker)
22 |  |  |  forward txPrepMsg to broker
24
25 Procedure handleTXPrepareAckMsg(txID, status) from origin
26 |  pEntry ← prepAckMap.get(txID)
27 |  if status = FALSE then  pEntry.setStatus(FALSE)
28 |  pEntry.removePendingAck(origin)
29 |  if pEntry.getPendingAcks() = ∅ then
30 |  |  ackMsg ← newPrepAckMsg(txID, pEntry.getStatus())
31 |  |  send ackMsg to pEntry.getOrigin()
32 |  else
   |  |                               // wait until all ACKs are receved.
```

are terminated; if not, the *prepare* round is paused until this is the case (Lines 2–3). Next, potential conflicts are identified by comparing, for all buffered publications, the routing behavior based on the transaction's state, `TXRS`, with the routing behavior based on the consistent routing state `RS`. In other words, a conflict occurs if a concurrent transaction ordered prior to the committing one has changed the routing state matching one of the publications. If a conflict is detected, a *status* flag in the `ackMap` entry for the prepare phase is set to (`FALSE`) (Line 12); otherwise, it is set to `TRUE`, i.e., no conflict. Ultimately, this status is reported to the TXC as part of the `prepAckMsg`.

Once the TXC receives a `prepAckMsg`, it starts the *commit* round: if the *status* flag is set to `TRUE`, the TXC sends a commit message, `cmtMsg(txID)`, else it sends an abort message, `abtMsg(txID)`. Both message types are flooded through the overlay and acknowledged in reverse direction. If a broker receives a `cmtMsg`, it merges `TXRS` with the stable routing state `RS`, truncates `TXContext`, and marks the transaction committed. If a broker receives an `abtMsg`, it discards `TXContext` and marks the transaction as aborted. Clients receiving a `cmtMsg`, deliver all buffered publications to the application, or discard them when receiving an `abtMsg`.

6.3.4 D-TX discussion

D-TX implements the distributed pub/sub transaction model with the strong ACID semantics defined in Section 6.2. A core concept of the approach is that every transaction operates on a dedicated snapshot, which is consistently taken from the state of a prior committed transaction. Consistency is achieved by ensuring that all operations are executed in the order specified by clients. This is guaranteed through an acknowledgment mechanism enforcing that every operation is completely processed before its subsequent operation. Atomicity and isolation are achieved by imposing a total order on transactions during initialization and an optimistic concurrency control mechanism, which identifies conflicts when the transaction is preparing to commit.

6.4 Static Transaction Service

We now describe the design of our S-TX approach to support static pub/sub transactions in tree-based overlays. In contrast to D-TX, here, the TXC has full knowledge about the transaction. For S-TX, we assume weak isolation: this means if two concurrent transactions do not operate on disjoint event spaces, still, routing states of brokers converge, but publications might be routed to concurrent subscribers. Again, we guarantee atomicity by adapting the 2-PC algorithm to ensure that publications are only delivered to the application when the transaction committed. Sequential consistency is realized by attaching a list of dependencies, referencing prior-ordered operations, to every operation, and evaluating at brokers whether dependencies have already been processed. Frequently used terminology in addition to the one for D-TX (cf. Table 6.3.1) is summarized in Table 6.4.1.

Table 6.4.1: S-TX notation.

`CRT`	client routing table
`RS`	stable routing state of broker, i.e., `SRT` and `PRT`
`processedOps`	maintains all processed operations
`opBuffer`	maintains operations with pending dependencies
`precedeMap`	maintains dependencies for each operation
`succeedMap`	references all operations depending on an operation
`overlayJoinMsg(clientID)`	overlay join message for client
`opMsg(txID,opID,op,` $\mathcal{D}$`)`	operation message for `op` $\in \mathcal{O}_{PS}$ depending on prior operation set $\mathcal{D}$
`tcm(`F_p`, tcmOPs)`	TCM operation, contains set of operations to trigger at receiver (`tcmOPs`)
`cmtMsg(txID,` $\mathcal{D}$`)`	commit message depending on $\mathcal{D}$
`abtMsg(txID,` $\mathcal{D}$`)`	abort message depending on $\mathcal{D}$

6.4.1 S-TX overview

Transaction processing in S-TX is realized in two phases: in the *operation* phase, the TXC generates all operations for the transaction and issues them to the system. `tcm` operations are used to send a subset of these operations to particular TXPs, which are intended to issue them as their own operations. Every operation message contains

a list of dependencies. This list describes all preceding operations and can also be statically computed by the TXC. Brokers evaluate whether all dependencies have been processed before processing the actual operation. In the *termination* phase, the transaction either commits, i.e., all publications are delivered to the application at TXPs, or aborts, i.e., compensating operations are sent and publications are discarded. Similar to D-TX, a broadcast is used to propagate the decision to brokers and clients.

6.4.2 S-TX client design

S-TX exposes the same API as D-TX (cf. Table 6.3.2) but uses a slightly different message format and transaction management. A `tcm` message has a payload field, `tcmOps`, that contains all operations a receiving TXP should issue for this transaction. In addition to `txID` and `opID`, every operation also has a payload field, `dependencies`, with information about prior operations.

Again, our use case can be implemented as sketched in Figure 6.1(b), where operations (1) is realized as *txTCM()* containing all other operations and dependencies in `tcmOps`.

A client still maintains a `TXManager` for every transaction it is involved in. The purpose of `TXManager` is to buffer incoming publications until the transaction finally commits, and to manage operations received from API calls that should be sent to the broker network, which includes attaching `opID` and `dependencies`. However, compared to D-TX, operations are no longer required to be buffered until the preceding operation has been acknowledged; instead, they are issued directly and the ordering is done at brokers based on the dependencies.

6.4.3 S-TX broker design & protocol

The core component of the S-TX broker is the transactional router, `TXRouter`, which processes all protocol messages in FIFO order. In S-TX, `TXRouter` maintains a single version of the routing state `RS` representing `SRT` and `PRT`. In addition, it maintains

a list of all processed operations (`processedOps`) and a map, `opBuffer`, buffering operations that are not processed because the corresponding dependencies have not been satisfied. Dependencies are represented using two data structures: the `precedeMap` maps an operation `opID` to IDs of all other operations, which `opID` depends on, i.e., all prior-ordered operations. The `succeedMap` maps an operation `opID` to all operations that depend on `opID`, i.e., all post-ordered operations.

Satisfiable dependencies – Before processing an operation, a broker verifies if all dependencies have been processed by checking `processedOps`. However, not every dependency is *satisfiable* at all brokers. (Un-) subscriptions are only processed by brokers on a path between the subscriber and clients with matching advertisements. To determine if a dependency is satisfiable and, hence, whether the operation must wait for the dependency to be processed, a broker validates if it is on such a path; this includes (1) checking whether the `PRT` of the broker contains a matching advertisement, and (2), whether the `lastHop` of this advertisement is different from the `lastHop` of the (subscription) dependency[2]. For this reason, dependencies do not only comprise the `opID` but also information about the issuing client (`senderID`) and about its filter predicates (`filter`):

$$\texttt{dependency} = (\texttt{opID},\ \texttt{senderID},\ \texttt{filter})$$

Client Routing Table – In addition, to reason about the origin of a depending operation, `TXRouter` maintains a *client routing table* (`CRT`), which is a list of [`clientID`, `nextHop`]-pairs. The `CRT` contains routing information for all clients connected to the broker network and thereby enables the broker to determine from which neighbor a dependency subscription will arrive.

After connecting to a broker, a new client issues an `overlayJoin` message. Similar to advertisements, this message is broadcast to all brokers in order to update the `CRT`.

Operation phase – Algorithm 6.4.1 describes the processing of a message `opMsg = (txID, opID, op,` $\mathcal{D}$`)` at brokers.

[2]It is not possible that a subscription and a matching advertisement arrive at a broker from the same neighbor because subscriptions are routed along the reverse path.

Algorithm 6.4.1: Pub/Sub operation processing in S-TX.

```
1  Procedure handleOpMsg(txID, opID, op, D)
2      D_s ← getSatisfiableDependencies (D)
3      forall d ∈ D_s do
4          if d ∉ processedOps then
5              precedeMap.put(opID, d)
6              succeedMap.put(d, opID)
7      if precedeMap.get(opID) = ∅ then
8          process(txID, opID, op)
9      else
10         opBuffer.add(opID, opMsg)
12
13 Procedure process(opMsg = (txID, opID, op))
14     RS.add(op)                     // (u)adv/(u)sub update routing state
15     destinations ← RS.getRoutingMatches(op)
16     forall dest ∈ destinations do
17         forward op to dest
18     markProcessed(opID)
20
21 Procedure markProcessed(opID)
22     processedOps.add(opID)
23     next ← {}
24     forall sID ∈ succeedMap do
25         precedeMap.get(sID).remove(opID)
26         if precedeMap.get(sID) = ∅ then
27             next ∪ sID
28     forall nID ∈ next do
29         process(nID, opBuffer(nID))
31
32 Procedure getSatisfiableDependencies(D)
33     D_s ← {}
34     forall d ∈ D | d ∈ {sub, usub} do
35         mAdvs ← RS.getMatchingAdvertisements(d.filter)
36         forall a ∈ mAdvs do
37             if a.lastHop = CRT.get(d.senderID) then
38                 D_s ∪ d
39     return D_s
41
```

First, the set of *satisfiable* dependencies for operation `opID` is calculated (Lines 2, 31–38); satisfiable dependencies that are not processed yet are stored in `precedeMap` and `succeedMap`. If not all dependencies are processed, i.e., not all corresponding operations have been processed, `opMsg` is buffered. Otherwise, the operation first updates the routing state `RS` and is then forwarded to neighboring brokers or clients according to `RS` (Lines 14–16). After processing operation `op`, it is marked as processed (Line 17). Procedure `markProcessed` includes removing the corresponding `opID` from the dependency sets of all post-ordered operations, `sID`, in its `succeedMap`. Whenever, an operation `sID` has no more unprocessed dependencies, i.e., its `precedeMap` is empty, `sID` is added to a set of operations that is processed next (Lines 20–28).

Termination phase – The intention of the *termination* phase is to either commit the transaction and deliver all publications, buffered at clients, to the application, or abort the transaction, discard buffered publications, and undo all changes applied to the routing state through compensating operations. Because S-TX assumes isolation to be managed at the application-level, an abort in S-TX does not occur due to conflicts among concurrent transactions but only due to an explicit command by the TXC.

Commit—to commit a transaction, the TXC issues a commit message, `cmtMsg = (txID,` $\mathcal{D}$`)`, to the system, where $\mathcal{D}$ contains dependencies to all operations of the transaction. `cmtMsg` is broadcast through the system; similar to the regular operation messages, brokers ensure that all dependencies are processed before forwarding `cmtMsg`. Clients and edge brokers receiving `cmtMsg` respond with a `cmtAckMsg`, which is aggregated at intermediary brokers to notify the TXC about the successful commit.

Abort—to abort a transaction, the TXC generates a sequence of compensating operations, $\mathcal{C}$ and issues them to the system followed by an abort message. For each operation changing the routing state in the transaction, $\mathcal{C}$ contains the inverse operation, e.g., for `sub`$(\mathcal{F}) \in \mathcal{T}_{PS}$, the operation `usub`$(\mathcal{F})$ is added to $\mathcal{C}$. To distribute the compensation operations to the TXPs that issued the original operation, the same `tcm` constructions are used as in $\mathcal{T}_{PS}$. Compensating operations are issued and processed in the same order as the original operations; their dependency lists

contain both references to the original operations and to prior-ordered compensating operations. The abort message, `abtMsg = (txID,` $\mathcal{D}$`)`, containing dependencies to all prior operations, is broadcast through the system. Clients receiving the `abtMsg` discard buffered publications and respond, just like edge brokers, with an `abtAckMsg`. These messages are again aggregated in the broker network to notify the TXC about the successful abort.

6.4.4 S-TX discussion

S-TX implements a relaxed variant of the pub/sub transaction model presented in Section 6.2 by only providing weak isolation. In general, the assumption in S-TX is that concurrent transactions operate on disjoint event spaces (application-level isolation), and, hence, no conflict detection is required. However, even if two concurrent transactions operate on overlapping spaces, routing states at brokers will converge, because both `SRT` and `PRT` can be considered conflict-free replicated data types (CRDT) [5] with *add()* as associative and commutative handler function. `add(sub, lastHop)`, for instance, adds a new entry to the `PRT` when processing a subscription, and `add(usub, lastHop)` adds and entry when processing the corresponding un-subscription. As both operations are idempotent, inconsistencies in the routing states are avoided in S-TX. Publications, however, might be delivered to unintended subscribers if transactions are not isolated at the application level through disjoint event spaces.

Also, different to D-TX is that the TXC is assumed to have complete knowledge about the transaction simplifying the implementation of consistency. Operation ordering is realized using dependencies attached to every operation, which can be calculated statically and before the transaction starts. Atomicity is realized similar to D-TX but does not require conflict detection.

6.5 Evaluation

6.5.1 Overview

We implemented D-TX and S-TX as extension to the PADRES enterprise event bus [41]. PADRES[3] is a content-based pub/sub implementation using a network of brokers to disseminate pub/sub operations as described in Section 6.1. We experimentally evaluated our implementations in an OpenStack cloud infrastructure, where every virtual machine (VM) was equipped with 4GB RAM, 2 vCPUs @ 2GHz, and Ubuntu 14.04.1 LTS. For our experiments, we implemented the instance dispatching scenario described in the beginning of Chapter 6, where a single transaction is comprised of 4 clients issuing five operations in total (cf. Figure 6.1(b)).

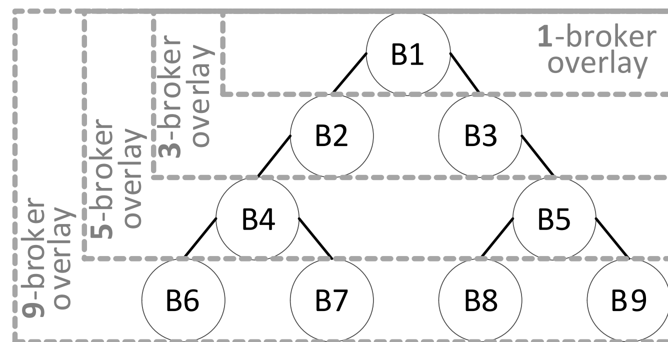


Figure 6.5.1: Broker overlays used in experiments.

We used different broker topologies (cf. Figure 6.5.1), multiple instantiations of the scenario (i.e., varying number of clients), and multiple degrees of concurrency (i.e., transactions a TXC manages concurrently). Every broker and every client was deployed on a separate VM and clients were uniformly distributed among brokers. To reason about the overhead transaction management introduces to pub/sub, we also implemented a baseline approach BL-WAIT, which does not provide isolation but only tries to do its best to achieve consistent operation ordering by manually introduced delays. It is important to note that BL-WAIT is not a viable, competing solution. We evaluated all three approaches w.r.t. *latency*, i.e., the time it takes to

[3]Code available under `https://github.com/MSRG/padres`

complete a single transaction, and *throughput*, i.e., how many transactions a system is able to process in a certain time.

6.5.2 Baseline implementation (BL-WAIT)

In the instance dispatching scenario (cf. Figure 6.0.1), operation ordering is critical to achieve the desired system behavior. In particular the subscription (Op. 3) and the unsubscription (Op. 4) must be completely processed at all brokers before the publication (Op. 5) is routed. D-TX and S-TX manage ordering by an acknowledgment mechanism or dependency checking, respectively. However, plain pub/sub does not provide such mechanisms; if the Environment (E) issues the second publication (Op. 5) right after the first publication (Op. 1), it is quite likely that it is routed to an unintended subscriber, i.e., the Dispatcher (D), and not an Agent (A). To prevent this and estimate the cost of ordering, we manually introduce a *wait* time between Op. 1 and Op. 5. To minimize *wait* in BL-WAIT, we approximate *wait* up to a delta $\Delta = 50ms$; the procedure is described in Algorithm 6.5.1. We do several experimental rounds and start running the scenario with a high value for $wait = 2000ms$; then, we gradually reduce/increase *wait* until we find a value that allows to execute a 1000 transactions correctly (i.e., the publication (Op. 5) is correctly routed) and, in addition, some prior run with a $wait_p = wait - \Delta$ has failed. Once, we fixed *wait*, we execute the scenario accordingly and measure latency and throughput. To take accurate measurements we use a timer at client E, which is started before sending Op. 1; when an Agent A receives the second publication (Op. 5), it sends an acknowledgment to E, which stops the timer.

6.5.3 Experiments

Base comparison – First, we compared the base performance of all approaches using an overlay with three brokers.

We used a single instantiation of the scenario, i.e., one Environment client (E) serving as TXC and one Dispatcher client (D), that dispatches workflow instances

Algorithm 6.5.1: Approximate *wait* in BL-WAIT.

```
1  Procedure approximateWait()
2      Δ = 50ms
3      r^c = ⊥                                   // result in current run.
4      r^l = ⊥                                   // result in last run.
5      wait^c = 2000 ms                          // wait time in current run.
6      wait^l = 0 ms                             // wait time in last run.
7      lastChange = wait^c − wait^l              // adaptation in last run.
8      δ = |lastChange| while ¬r^c ∧ δ > Δ do
9          r^c = runExperiment(w^c)
10         δ = |wait^l − wait^c|
                                                 // decrease wait.
11         if r^c then
12             if r^l then
13                 γ = −|lastChange|
14             else
15                 γ = −⌈|lastChange|⌉ × 0.5
                                                 // increase wait.
16         else
17             if r^l then
18                 γ = +⌈|lastChange|⌉ × 0.5
19             else
20                 γ = +|lastChange|
21         wait^l = wait^c
22         r^l = r^c
23         wait^c = wait^c + γ
24         lastChange = γ
26     return
```

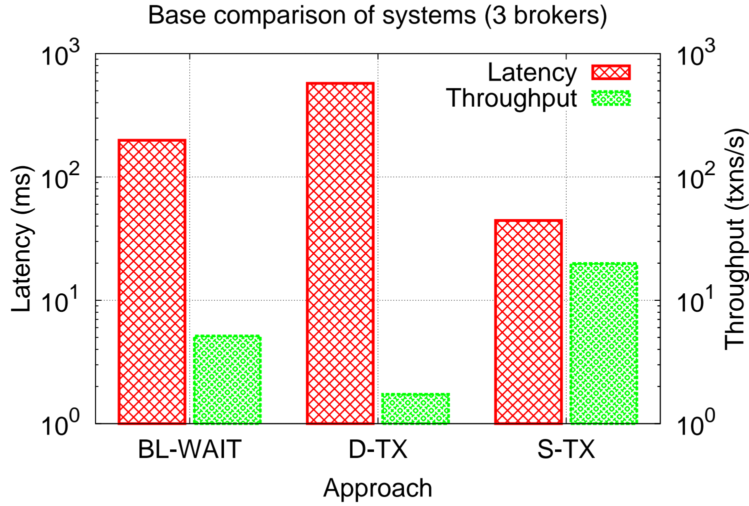
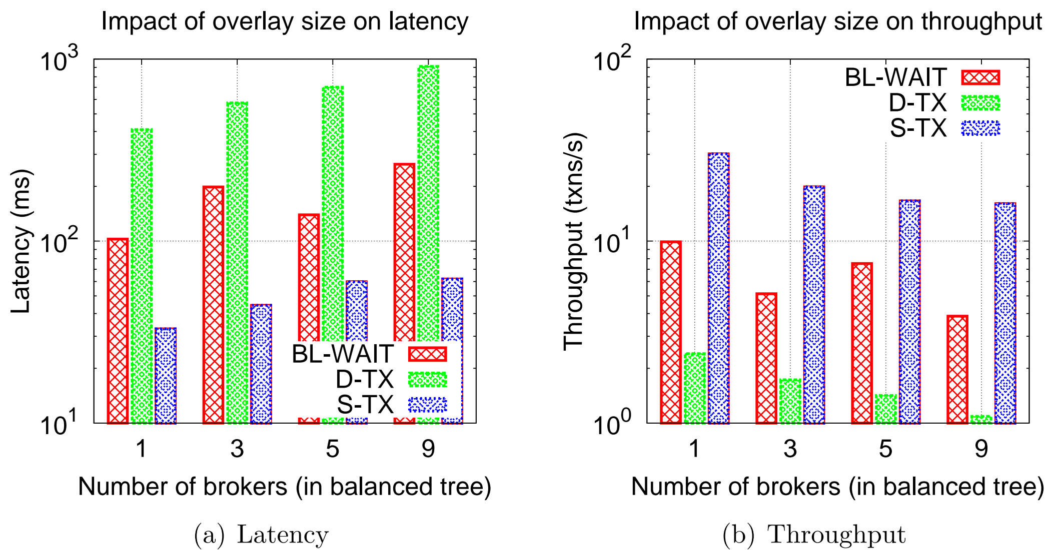


Figure 6.5.2: Base comparison.

in an alternating fashion to two Agent clients (A1, A2). All clients have been randomly but uniformly connected to all brokers. In every experiment, we executed 1000 transactions (i.e., dispatched 1000 workflow instances). All transactions were executed sequentially by the TXC. The results obtained for the three approaches are depicted in Figure 6.5.2. Since transactions were executed sequentially, i.e., the TXC waited for the completion of one transaction before starting the next, throughput and latency are directly related by $throughput \times latency = 1s$. Unsurprisingly, D-TX (570 ms averaged latency) performs worse than BL-WAIT(190 ms); the reason for the difference is the increased message overhead for ordering, which requires acknowledgments for every operation, and for isolation, which requires three broadcast rounds in total (one for initialization and two for termination). More surprising is that S-TX (~ 20 $txns/s$) beats the performance of BL-WAIT (~ 5 $txns/s$). To some extend the huge speedup ($\sim$ 4X) can be explained by the imprecision introduced with the wait accuracy through Δ. Another part of speedup results from the reduced transmission time for the second publication (Op. 5), which was buffered in the network in S-TX and not at client E as in BL-WAIT. Most importantly, however, is that in BL-WAIT, we had to fix parameter $wait$ conservatively so that the desired correct output was achieved in every single transaction. The graph in Figure 6.5.2 shows the average latencies and in the experiments on S-TX, we observed that some transactions terminated much faster than the average. Similar, in our parametrization

runs for BL-WAIT, we found that for some of the transactions correct processing was achieved with less wait time. In these cases, *wait* introduced unnecessary latency, which, however, is important to ensure that all transactions terminate correctly. In S-TX, to the contrast, the wait time was minimal in every transaction as the publication could be directly processed once the dependencies were fulfilled.

Impact of overlay size on performance – In another experiment, we analyzed the impact of the broker overlay on the performance. We used a single instantiation of the instance dispatching scenario (i.e., four clients), executed 1000 transactions sequentially, and compared four different overlay networks (shown in Figure 6.5.1). Again the clients have been distributed randomly and uniformly among brokers: for the single broker overlay, all clients were connected to the single broker, for the three-broker overlay, one broker was connected with two clients, and for the remaining overlays some brokers had no direct client connection. Throughout our experiments, we varied the allocation of clients to brokers. The results represent the averages of all experiment runs and are depicted in Figure 6.5.3.



(a) Latency

(b) Throughput

Figure 6.5.3: Performance impact of broker overlay size.

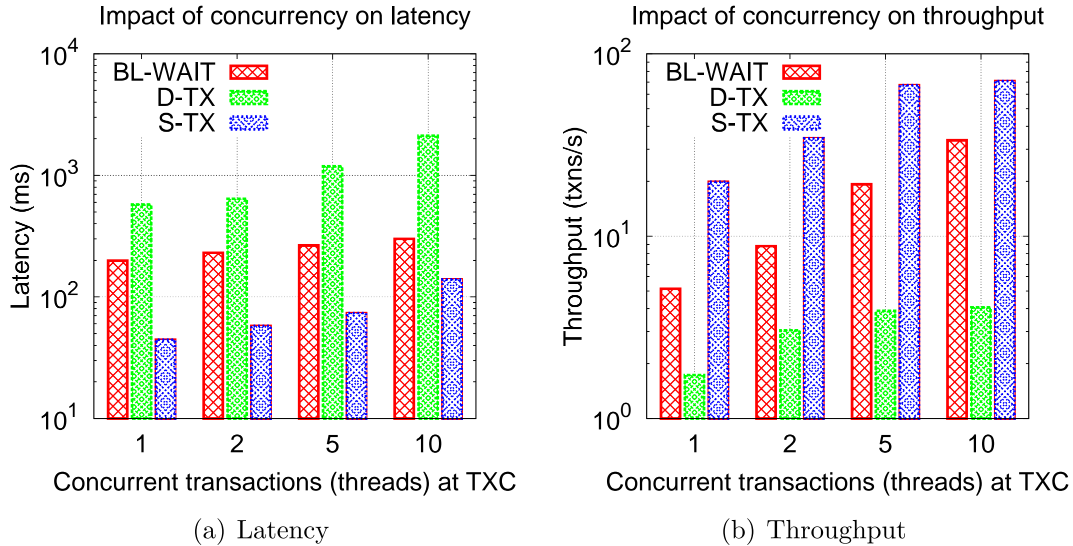
Basically, the base performance relation among all three approaches is confirmed (cf. also Figure 6.5.2): BL-WAIT is faster than D-TX in all overlay settings and S-TX always beats BL-WAIT. However, with increasing overlay sizes, the performance

of all three approaches declines. For instance, the average latency of D-TX for a single broker, $\sim 410ms$, is more than double that of nine brokers, $\sim 911ms$, resulting in ~ 2.5X performance degradation. This decline can also be observed in BL-WAIT (~ 2.6X). In S-TX, the degradation is a little less (~ 1.9X). In general, the performance drop with increasing overlays can be explained by the increased communication effort resulting from additional hops every message must take. D-TX is particularly affected for two reasons: first, also the acknowledgments must traverse more hops, and second, the broadcast rounds in initialization and termination phase require more hops. S-TX, to the contrast, requires fewer broadcasts and no acknowledgments; hence, it is less sensitive to the network size and the latency of $\sim 60ms$ with nine brokers is still very good.

Impact of concurrency on performance – In this experiment, we investigated the impact of concurrency on performance. With concurrency, we refer to the number of parallel transactions (threads) initiated by the TXC. We varied the number of threads between, $1, 2, 5$, and 10. Again, we used a single instantiation of the scenario (i.e., four clients), fixed the overlay to three brokers, and executed 1000 transactions. Note, that there is no conflicts among transactions, as the TXC dispatches different workflow instances in every transaction, i.e., the event spaces do not overlap.

The results are depicted in Figure 6.5.4. Again, S-TX performs better than BL-WAIT, which beats D-TX. As expected, for all three approaches, the latencies are increasing with increased concurrency due to concurrent processing at brokers and clients. While BL-WAIT is comparably stable (latency increases by 50% for 10 concurrent threads compared to sequential execution), the latencies for S-TX (3X) and D-TX (4X) grow faster. In contrast, concurrency also has a positive effect on throughput as brokers spend less time idling: for BL-WAIT throughput increases by 7X, for S-TX by 3.7X and for D-TX by 2.3X. Although, in absolute numbers, S-TX reveals the best performance, its concurrency behavior is worse compared to BL-WAIT because of the management overhead for the termination phase. For similar reasons, also the concurrency behavior of D-TX is worse compared to BL-WAIT. In conclusion, concurrency increases the throughput but has to be paid with latency.

Impact of client quantity on performance – In a last experiment, we scaled the

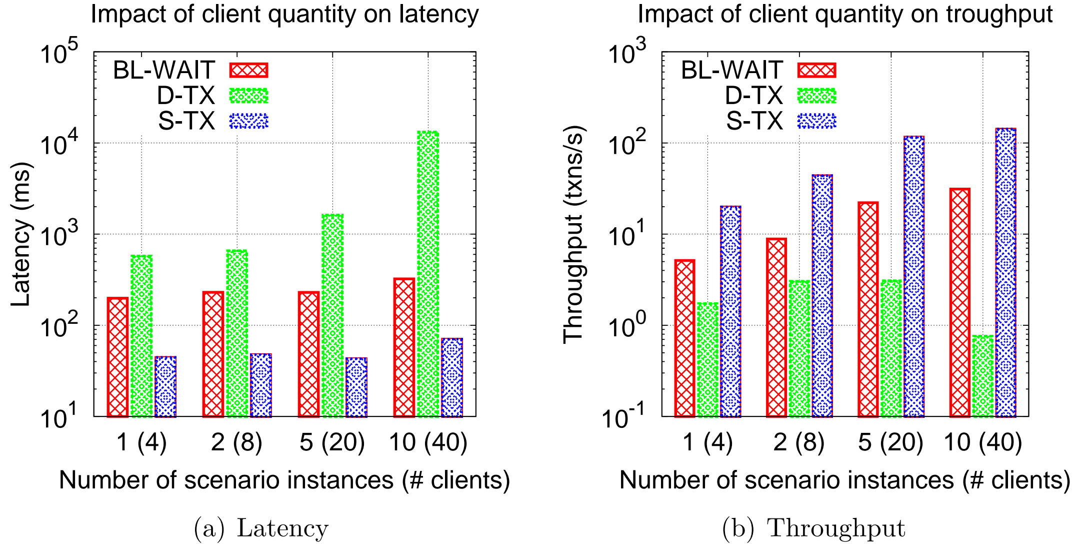


(a) Latency (b) Throughput

Figure 6.5.4: Performance impact of concurrency.

number of clients that concurrently execute transactions and analyze the performance impact. To this end, we varied the number of instantiations of the instance dispatching scenario from a single instance (i.e., 4 clients) to two, five, and ten instances (i.e., 40 clients). Each instantiation dispatches instances for a different workflow, so the event spaces of individual transactions do not overlap. We used a fixed overlay of three brokers and at every TXC, transactions are sequentially executed. The results are depicted in Figure 6.5.5.

Again, S-TX performs better than BL-WAIT and D-TX for all configurations. In addition, the scaling behavior for S-TX and BL-WAIT is pretty similar: in both approaches, the latencies increase by only 50% but the throughput increases by 7X when scaling up to 40 clients. Compared to scaling at a single client (cf. experiments on concurrency and Figure 6.5.4), the two approaches perform better, which gives evidence that part of the latency results from the concurrency at clients. The scaling behavior of D-TX, however, is not as good. While throughput increases up to 20 clients, the performance begins to drop again for 40 clients. The main reason for this is the isolation management of D-TX, which requires agreement on a consistent snapshot during initialization and, more important, maintaining the commit order during termination. Adherence to the commit order may delay some transactions



(a) Latency

(b) Throughput

Figure 6.5.5: Performance impact of client quantity.

that are ready to commit because they have to wait for a prior-ordered transaction to terminate.

6.5.4 Discussion

We evaluated D-TX and S-TX by comparing them to a baseline solution BL-WAIT, with which we tried to get an estimate on the plain performance of the underlying pub/sub system and an idea about the additional cost transaction management introduces. Note that BL-WAIT is not really a competing solution, as the parameter *wait* must be carefully chosen, depends on the current system, and requires a set of prior configuration rounds, which is impractical. In our experiments, we showed that S-TX efficiently guarantees consistency according to an application specific order and even beats this baseline in all experiments. Isolation, however, as guaranteed by D-TX is costly and practicable only for smaller settings.

6.6 Summary

In this chapter, we formalized pub/sub transactions as a sequence of pub/sub operations that are to be atomically processed by a distributed pub/sub system isolated from concurrent transactions. In our model, we allow that publications by one client can trigger further operations by different clients—forming truly distributed transactions. Based on the a priori knowledge a transaction coordinator (TXC) has, we provide two approaches implementing our model: D-TX assumes no prior knowledge on operations by other transaction participants (TXP). Acknowledgments enable consistent operation ordering, and snapshot isolation enables serializability. S-TX relaxes the above assumptions: isolation is considered to be managed at the application level and the TXC has full knowledge about the transaction. However, even if concurrent transaction are not perfectly isolated at the application level, S-TX achieves convergence of the routing states at brokers. In S-TX, consistency is achieved through dependency checking among operations at brokers. Both approaches adopt 2-PC to provide atomicity.

Our experimental evaluation showed that isolation and the uncertainty about operations renders D-TX costly and suitable only for smaller configurations. In contrast, S-TX introduces no significant overhead, which has been proven by comparing to a baseline pub/sub mimicking the guarantees.

Chapter 7

Conclusions

7.1 Summary

For many organizations, business process and workflow automation is becoming an important means to reduce process execution time, improve resource utilization, and enhance service quality, and, thereby, increase the overall business profitability. Recent trends and prevailing circumstances, however, prevent technology from unleashing its full potential: first, the ever-increasing amount of knowledge-intense work requires ad-hoc and flexible business processes, which only few workflow management systems (WFMSs) nowadays support. Next, and orthogonal to this, many business processes include data and process fragments from different organizations; current WFMSs do not consider this distribution appropriately, either resulting in unnecessary data transport, or complicating compliance with data protection law. In addition, the increasing scale of business processes, mostly in terms of concurrent instances, is a challenge for the designers of WFMSs.

Data-centric workflows constitute a promising substrate for addressing the above challenges because they unify data and process modeling. Commonly, data-centric approaches built on top of an information model that describes all relevant aspects of the process, including application data and status information. The control-

flow is then expressed over declarative rules to evaluate the current state of the information model, evolve the process to a new state, and trigger relevant activities. The implicit rule formulation is less rigid compared to traditional activity-centric modeling approaches and is a cornerstone in supporting flexibility. At the same time, the publish/subscribe (pub/sub) paradigm and corresponding middleware implementations provide a convenient basis for the implementation of WFMSs. The loose coupling properties of publish/subscribe together with its event-based nature enable the integration of a WFMS in its business environment but also support the design of distributed WFMS solutions.

As a prominent representative of data-centric workflows, in this work, we focused on the Guard-Stage-Milestone (GSM) meta-model for specifying the lifecycle of business artifacts. GSM provides a rich theoretical foundation; its operational semantics are based on the incremental firing of ECA-like rules, called Prerequisite-Antecedent-Consequent (PAC) rules in GSM, in the order implied by a specific dependency graph, called Polarized Dependency Graph (PDG) in GSM. Most importantly, GSM also forms the basis for the operational semantics of the Case Management Model and Notation (CMMN)—the current standard for flexible business process modeling.

To provide an adequate automation support for GSM taking data distribution and related constraints, such as data-locality, into consideration, we first developed the theoretical foundation for a safe distribution and parallel execution of data-centric workflows over the distributed publish/subscribe abstraction. We developed a polynomial-time mapping of GSM into pub/sub by fragmenting the GSM workflow into a set of data-access and a set of control-flow components. To integrate the GSM semantics into pub/sub components, we redefined publications and subscriptions together with their matching, consumption, and notification policies. We proved the correctness of our mapping through an equivalence of reachable system snapshots and we also proved the hardness of the optimal workflow distribution problem over the pub/sub abstraction. To provide a solution to this distribution problem, we employed a greedy algorithm with a constant factor approximation.

Based on our theoretical results, we designed a corresponding conceptual architecture and a geo-scale distributed WFMS. Our system provides the automated execution of

flexible business processes specified in GSM and geographically scattered across different organizational units, while supporting locality of process- and data fragments. Our management infrastructure is fully distributed and based on Workflow Units (WFUs). WFUs are independent system components and represent the unit of distribution to either manage individual PAC rules or attributes from the information model. We presented two mapping implementations of GSM into the WFU representation: the baseline mapping (BLM) is directly based on our pub/sub formalization. The optimized context-aware mapping (CAM) results from the insights we obtained from implementing BLM. In contrast to BLM, CAM considers the type of external events to reduce the computational overhead and network communication delay that are further magnified at geo-scale. Our experimental evaluation in a cloud environment showed that both mappings are scalable but CAM outperforms BLM. In particular, CAM reveals its strengths for sequential task patterns in workflows, while task splits/merges are more costly.

To support safe horizontal scaling of WFMSs integrated over pub/sub, which requires replication of the workflow engine for individual instances and dispatching, we presented an approach for multi-client transactions in distributed pub/sub middleware. We first formalized pub/sub transactions as a sequence of pub/sub operations that are to be atomically processed but isolated from any concurrent transactions, while we allow that publications by one client can trigger further operations by different clients. Based on the a priori knowledge of the transaction coordinator (TXC), we present two approaches implementing our model: D-TX assumes no prior knowledge on operations by other transaction participants (TXP). Acknowledgments enable consistent operation ordering and sequential consistency, and snapshot isolation enables serializability. S-TX relaxes the above assumptions: isolation is considered to be managed at the application level and the TXC has full knowledge about the transaction. Even if concurrent transaction are not perfectly isolated at the application level, S-TX achieves convergence of the routing states at brokers. In S-TX, consistency is achieved through dependency checking among operations at brokers. Both approaches adopt 2-PC to provide atomicity. Our experiments showed that isolation and the uncertainty about operations renders D-TX costly and suitable only for smaller configurations. In contrast, S-TX introduces no significant overhead, which has been proven by comparing to a baseline pub/sub mimicking the guarantees.

7.2 Future work

Future work related to the theory on fragmentation and distribution of flexible and data-centric workflows includes (i) investigating refinements and optimizations of the mapping into the pub/sub abstraction: for instance, to reduce the overhead of bookkeeping, one could think of evaluating different, possibly heuristic, strategies; in addition, to further increase parallelism in execution, an approach to leverage data independences between different B-steps and a relaxation of the requirement for a global external event-queue could be considered; (ii) increasing the generality of our approach: for instance, by extending the state model from two-valued (i.e., Boolean) states to supporting arbitrary multi-valued state machines for handling different transactional scenarios; (iii) supporting dynamic changes in workflows: for instance, through *stage factories* that allow for on-the-fly creation and incorporation of additional stage instances to enable ad-hoc workflow adaptations at runtime.

In the same spirit, future work related to our geo-distributed WFMS could consider flexibility at runtime, where it is necessary to adapt the workflow model; this, in turn, requires online adaptations to the engine, i.e., as instances are executed in parallel. Changes to the execution engine can be seen as the addition, modification, or removal of WFUs resulting from a recompilation of the updated workflow model. The main challenges are to maintain the semantic correctness of active instances and to avoid disrupting their execution, e.g., by pausing the execution for an inadequate amount of time. A potential solution could be to introduce WFU versioning, i.e., a new version for each adaptation. Active instances execute on older versions until they reach a state that allows them to safely migrate to the new version. Orthogonal to this, supporting privacy requires a location-aware deployment of WFUs to geo-distributed computing infrastructures. To automatically calculate such deployments, the process model must be enriched with additional information. First, information about all organizations and their computing infrastructures is required. Second, it is necessary to model which process fragments (i.e., data and rules) belong to which organizations. The main challenge is to find deployments that adhere to the locality constraints on one side, but also respect a set of other constraints, such as minimizing wide-area communication, on the other side.

Future work related to our multi-client transaction approach would mostly address the realization of the isolation property, which currently introduces considerable overhead in terms of messages and coordination that is necessary among brokers in D-TX. Potentially, the adaptation of lower isolation levels, such as *read committed* or *repeated reads*, to the distributed pub/sub model could provide a starting point for more efficient algorithms. Considering different isolation levels would, however, require to first investigate their implications on the application behavior and, in general, study the requirements of different transactional pub/sub scenarios. Another approach to facilitate isolation handling more from the application side instead of only using broker-internal mechanisms, could require the TXC to announce a transaction space before starting the transactions. The transaction space would be a subspace of the complete event space and avoid transaction ordering and conflict detection among concurrent and isolated transactions. Orthogonal to these considerations, general-purpose distributed algorithms could be considered to deal with safe state replication across brokers. For instance, the Paxos [53] protocol could be directly applied to provide atomicity in operation replication.

List of Figures

1.1.1 Referral chain in patient care 5
1.1.2 High-level overview of patient care business processes 7

3.1.1 Illustration of GSM B-step. 39

4.1.1 ”Design-to-order” business process modeled in GSM. 49
4.1.2 Polarized dependency graph (PDG) for “Design-to-order” workflow. 52
4.3.1 High-level illustration of subscription flow. 60
4.4.1 Consumption policy state transition. 69
4.6.1 Illustration of subscription assignment. 79

5.1.1 Patient care process in CMMN. 87
5.1.2 Patient care process in GSM. 88
5.1.3 Lifecycle for core CMMN elements and corresponding GSM representation 91
5.2.1 Distributed GSM workflow architecture. 92
5.2.2 WFU partitioning of global data model. 94
5.3.1 Example of BLM mapping for PAC rule. 97
5.3.2 Example of BLM mapping for status attribute. 98
5.3.3 Example of CAM mapping for PAC rule. 100
5.3.4 Example of CAM mapping for status attribute. 102
5.5.1 Evaluation of Patientcare business process. 107
5.5.2 Evaluation of BLM vs. CAM: impact of model type on performance. 108

5.5.3 Evaluation of BLM vs. CAM: scaling analysis. 109
5.5.4 Evaluation CAM Mapping: impact of ePDG size. 110

6.0.1 Use case: instance dispatching in WFMS. 114
6.2.1 Example for sequential consistency. 121
6.3.1 Internal broker architecture in D-TX. 126
6.5.1 Broker overlays used in experiments. 139
6.5.2 Base comparison. 142
6.5.3 Performance impact of broker overlay size. 143
6.5.4 Performance impact of concurrency. 145
6.5.5 Performance impact of client quantity. 146

List of Tables

3.1.1 Templates for PAC rules associated with a GSM model. 37

4.1.1 Sentries associated with guards. 50
4.1.2 Sentries associated with milestones. 50
4.1.3 Complete set of PAC rules for the “Design-to-order” workflow. . . 51
4.1.4 Excerpt of PAC rules for “Design-to-order” workflow. 51

5.1.1 Excerpt of PAC Rules for “Patient care” workflow. 90
5.5.1 Synthetic workflow models for evaluation. 106

6.3.1 D-TX notation. 123
6.3.2 Client interface. 124
6.4.1 S-TX notation. 133

List of Algorithms

5.4.1 Process workflow event at WF Agent. 104
5.4.2 Generate notifications. 105

6.3.1 Initialization phase in D-TX. 128
6.3.2 Pub/Sub operation processing in D-TX. 129
6.3.3 Prepare phase in D-TX. 131
6.4.1 Pub/Sub operation processing in S-TX. 136
6.5.1 Approximate *wait* in BL-WAIT. 141

Bibliography

[1] S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML Project: An Overview. *The VLDB Journal*, 17(5):1019–1040, 2008.

[2] S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu. The AXML Artifact Model. In *Proceedings of the 16th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 11–17, 2009.

[3] S. Abiteboul, P. Bourhis, B. Marinoiu, and A. Galland. AXART: Enabling Collaborative Work with AXML Artifacts. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1-2):1553–1556, 2010.

[4] A. Adi and O. Etzion. Amit - The Situation Manager. *The VLDB Journal*, 13(2):177–203, 2004.

[5] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro. Cure: Strong Semantics Meets High Availability and Low Latency. In *Proceedings of the 36th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 405–414, 2016.

[6] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. E. Abbadi, and M. Kamath. Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *Proceedings of the IFIP working conference on information systems development for decentralized organizations (IFIP)*, pages 1–18, 1995.

[7] Amazon. Amazon's Corporate IT Migrates Business Process Management to the Amazon Web Services Cloud. `https://media.amazonwebservices.com/AWS_Amazon_BPM_Migration.pdf`, 2011.

[8] A. Andersen, K. Y. Yigzaw, and R. Karlsen. Privacy preserving health data processing. In *2014 IEEE 16th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 225–230, 2014.

[9] Apache. ActiveMQ. `http://activemq.apache.org/`, 2017.

[10] S. Appel, S. Frischbier, T. Freudenreich, and A. Buchmann. Event Stream Processing Units in Business Processes. In *Proceedings of the 11th International Conference on Business Process Management (BPM)*, pages 187–202, 2013.

[11] A. Avanes and J. C. Freytag. Adaptive Workflow Scheduling under Resource Allocation Constraints and Network Dynamics. *Proceedings of the VLDB Endowment (PVLDB)*, 1(2):1631–1637, 2008.

[12] T. Bauer and P. Dadam. Efficient Distributed Workflow Management Based on Variable Server Assignments. In *Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 94–109, 2000.

[13] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[14] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4), 2007.

[15] D. Boaz, T. Heath, M. Gupta, L. Limonad, Y. Sun, R. Hull, and R. Vaculín. The ACSI Hub: A Data-centric Environment for Service Interoperation. In *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM)*, page 11, 2014.

[16] D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data-aware process analysis: a database theory perspective. In *Proceedings of the 32nd ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 1–12, 2013.

[17] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-area Event Notification Service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[18] F. Casati and A. Discenza. Modeling and Managing Interactions among Business Processes. *Journal of Systems Integration*, 10(2):145–168, 2001.

[19] A. Chan. Transactional Publish/Subscribe: The Proactive Multicast of Database Changes (Abstract). In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, page 521, 1998.

[20] T. Chao, D. Cohn, A. Flatgard, S. Hahn, M. Linehan, P. Nandi, A. Nigam, F. Pinel, J. Vergo, and F. Wu. Artifact-Based Transformation of IBM Global Financing. In *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*, pages 261–277. Springer Berlin Heidelberg, 2009.

[21] D. Cohn, P. Dhoolia, F. T. Heath, F. Pinel, and J. Vergo. Siena: From PowerPoint to Web App in 5 Minutes. In *Proceedings of 6th International Conference on Service-Oriented Computing (ICSOC)*, pages 722–723, 2008.

[22] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The Complexity of Multiway Cuts (Extended Abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 241–251, 1992.

[23] E. Damaggio, R. Hull, and R. Vaculín. On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard-Stage-Milestone Lifecycles. *Information Systems*, 38(4):561–584, 2013.

[24] M. Dumas. On the Convergence of Data and Process Engineering. In *Proceedings of 15th International Conference on Advances in Databases and Information Systems (ADBIS)*, pages 19–26, 2011.

[25] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management.* Springer, 2013.

[26] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley, 2005.

[27] European Commission. The EU-U.S. Privacy Shield — Adequacy Decision. `http://ec.europa.eu/justice/data-protection/files/privacy-shield-adequacy-decision_en.pdf`, July 2016.

[28] S. Ganesan, Y. Yoon, and H. Jacobsen. NIÑOS Take Five: The Management Infrastructure for Distributed Event-Driven Workflows. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System (DEBS)*, pages 195–206, 2011.

[29] Great Britain. Data Protection Act. 1998.

[30] GS1. Global Data Synchronisation Network (GDSN) — Operating Roadmap for GS1 Data Excellence. `http://www.gs1.org/docs/gdsn/support/gdsn_roadmap.pdf`, 2013.

[31] F. T. Heath, D. Boaz, M. Gupta, R. Vaculín, Y. Sun, R. Hull, and L. Limonad. Barcelona: A Design and Runtime Environment for Declarative Artifact-Centric BPM. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC)*, pages 705–709, 2013.

[32] P. Hens, M. Snoeck, G. Poels, and M. D. Backer. Process Fragmentation, Distribution and Execution Using an Event-based Interaction Scheme. *Journal of Systems and Software*, 89:170–192, 2014.

[33] J. C. Hill, J. C. Knight, A. M. Crickenberger, and R. Honhart. Publish and Subscribe with Reply. Technical Report TR CS-2002-32, University of Virginia, 2002.

[34] C. Houy, P. Fettke, P. Loos, W. M. P. van der Aalst, and J. Krogstie. Business Process Management in the Large. *Business & Information Systems Engineering (BISE)*, 3(6):385–388, 2011.

[35] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen. Transactional Mobility in Distributed Content-Based Publish/Subscribe Systems. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 101–110, 2009.

[36] R. Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In *On the Move to Meaningful Internet Systems (OTM)*, pages 1152–1163, 2008.

[37] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In *7th International Workshop Web Services and Formal Methods (WS-FM)*, pages 1–24, 2010.

[38] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculín. Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pages 51–62, 2011.

[39] R. Hull and J. Su. NSF Workshop on Data-Centric Workflows. `http://dcw2009.cs.ucsb.edu/report.pdf`, 2009.

[40] IBM. MQSeries. `http://www-03.ibm.com/software/products/en/ibm-mq/`, 2017.

[41] H.-A. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. IGI Global, 2010.

[42] H. Jafarpour, B. Hore, S. Mehrotra, and N. Venkatasubramanian. Subscription Subsumption Evaluation for Content-based Publish/Subscribe Systems. In *Proceedings of the 9th International Middlware Conference*, pages 62–81, 2008.

[43] B. Javadi, M. Tomko, and R. O. Sinnott. Decentralized Orchestration of Data-centric Workflows Using the Object Modeling System. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 73–80, 2012.

[44] M. Jergler, H.-A. Jacobsen, M. Sadoghi, R. Hull, and R. Vaculín. Safe Distribution and Parallel Execution of Data-centric Workflows over the Publish/Subscribe Abstraction. In *32nd IEEE International Conference on Data Engineering (ICDE)*, pages 1498–1499, 2016.

[45] M. Jergler, M. Sadoghi, and H.-A. Jacobsen. D2WORM: A Management Infrastructure for Distributed Data-centric Workflows. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1427–1432, 2015.

[46] M. Jergler, M. Sadoghi, and H.-A. Jacobsen. Geo-Distribution of Flexible Business Processes over Publish/Subscribe Paradigm. In *Proceedings of the 17th ACM International Middleware Conference*, pages 15:1–15:13, 2016.

[47] M. Jergler, K. Zhang, and H.-A. Jacobsen. Multi-client Transactions in Distributed Publish/Subscribe Systems. *Submitted to 18th ACM International Middleware Conference*, 2017.

[48] G. Keller, M. Nüttgens, and A. Scheer. *Semantische Prozessmodellierung auf der Grundlage "ereignisgesteuerter Prozessketten (EPK)"*. Institut für Wirtschaftsinformatik Saarbrücken: Veröffentlichungen des Instituts für Wirtschaftsinformatik. Inst. für Wirtschaftsinformatik, 1992.

[49] M. Kim, A. Mohindra, V. Muthusamy, R. Ranchal, V. Salapura, A. Slominski, and R. Khalaf. Building scalable, secure, multi-tenant cloud services on IBM Bluemix. *IBM Journal of Research and Development*, 60(2-3), 2016.

[50] E. Kucukoguz and J. Su. On Lifecycle Constraints of Artifact-Centric Workflows. In *7th International Workshop on Web Services and Formal Methods (WS-FM)*, pages 71–85, 2010.

[51] J. Kunchala, J. Yu, and S. Yongchareon. A Survey on Approaches to Modeling Artifact-Centric Business Processes. In *WISE Workshops*, volume 9051 of *Lecture Notes in Computer Science*, pages 117–132, 2014.

[52] V. Künzle and M. Reichert. PHILharmonicFlows: Towards a Framework for Object-aware Process Management. *Journal of Software Maintenance*, 23(4):205–244, 2011.

[53] L. Lamport. The Part-Time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[54] G. Li and H.-A. Jacobsen. Composite Subscriptions in Content-based publish/Subscribe Systems. In *Proceedings of the 6th International Middleware Conference*, pages 249–269, 2005.

[55] G. Li, V. Muthusamy, and H.-A. Jacobsen. A Distributed Service-oriented Architecture for Business Process Execution. *ACM Transactions on the Web (TWEB)*, 4(1):2:1–2:33, 2010.

[56] C. Liebig, M. Malva, and A. P. Buchmann. Integrating Notifications and Transactions: Concepts and X2TS Prototype. In *Proceedings of the 2nd International Workshop on Engineering Distributed Objects (EDO)*, pages 194–214, 2000.

[57] C. Liebig and S. Tai. Middleware Mediated Transactions. In *3rd International Symposium on Distributed Objects and Applications*, pages 340–350, 2001.

[58] L. Limonad, D. Boaz, R. Hull, R. Vaculin, and F. Heath. A Generic Business Artifacts Based Authorization Framework for Cross-Enterprise Collaboration. In *Proceedings of the SRII Global Conference*, 2012.

[59] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don'T Settle for Eventual: Scalable Causal Consistency for Wide-area Storage with COPS. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 401–416, 2011.

[60] H. D. Man. Case Management: Cordys Approach, 2009.

[61] J. Manyika, M. Chui, J. Bughin, R. Dobbs, P. Bisson, and A. Marrs. *Disruptive technologies: Advances that will transform life, business, and the global economy.* The McKinsey Global Institute, 2013.

[62] M. Marin, R. Hull, and R. Vaculín. Data Centric BPM and the Emerging Case Management Standard: A Short Survey. In *Proceedings of the 10th International Conference on Business Process Management (BPM) - Workshops*, pages 24–30, 2012.

[63] A. Marrella, M. Mecella, A. Russo, S. Steinau, K. Andrews, and M. Reichert. A Survey on Handling Data in Business Process Models (Discussion Paper). In *23rd Italian Symposium on Advanced Database Systems (SEBD)*, pages 304–311, 2015.

[64] A. Michlmayr and P. Fenkam. Integrating Distributed Object Transactions with Wide-Area Content-Based Publish/Subscribe Systems. In *In Proceedings of the 25th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 398–403, 2005.

[65] H. Mili, G. Tremblay, G. B. Jaoude, É. Lefebvre, L. Elabed, and G. E. Boussaidi. Business process modeling languages: Sorting through the alphabet soup. *ACM Computing Surveys (CSUR)*, 43(1):4, 2010.

[66] P. Muth, D. Wodtke, J. Weißenfels, A. K. Dittrich, and G. Weikum. From Centralized Workflow Specification to Distributed Workflow Execution. *Journal of Intelligent Information Systems*, 10(2):159–184, 1998.

[67] M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing Execution of Composite Web Services. In *Proceedings of the 19th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) 2004*, pages 170–187, 2004.

[68] K. Ngamakeur, S. Yongchareon, and C. Liu. A Framework for Realizing Artifact-Centric Business Processes in Service-Oriented Architecture. In *Proceedings of the 17th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 63–78, 2012.

[69] A. Nigam and N. S. Caswell. Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[70] OASIS. Web Services Business Process Execution Language v2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`, April 2007.

[71] Object Management Group (OMG). Unified Modeling Language (UML) Version 2.0). `http://www.omg.org/spec/UML/2.0/`, July 2005.

[72] Object Management Group (OMG). Business Process Model and Notation (BPMN) Version 2.0. `http://www.omg.org/spec/BPMN/2.0/`, Januar 2011.

[73] Object Management Group (OMG). Case Management Model and Notation (CMMN) Version 1.0. `http://www.omg.org/spec/CMMN/1.0`, 2014.

[74] E. S. Ogasawara, J. Dias, V. Silva, F. S. Chirigati, D. de Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16):2327–2341, 2013.

[75] M. Ould. *Business Processes: Modelling and Analysis for Re-Engineering and Improvement*. Wiley, 1995.

[76] P. R. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS), Workshops*, pages 611–618, 2002.

[77] Rabbit Technologies. RabbitMQ. `https://www.rabbitmq.com/`, 2017.

[78] Y. Raz. The principle of commitment ordering, or guaranteeing serializability in a heterogeneous environment of multiple autonomous resource mangers using atomic commitment. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB)*, pages 292–312, 1992.

[79] G. Redding, M. Dumas, A. H. M. ter Hofstede, and A. Iordachescu. Modelling Flexible Processes with Business Objects. In *Proceedings of IEEE Conference on Commerce and Enterprise Computing (CEC)*, pages 41–48, 2009.

[80] Redis Labs. Redis. `https://redis.io/`, 2017.

[81] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 1113–1114, 2005.

[82] M. Reichert, S. Rinderle-Ma, and P. Dadam. *Flexibility in Process-Aware Information Systems*, pages 115–135. Springer Berlin Heidelberg, 2009.

[83] M. Reichert and B. Weber. *Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies*. Springer Science & Business Media, 2012.

[84] M. Rosa, W. M. P. van der Aalst, M. Dumas, and F. P. Milani. Business Process Variability Modeling: A Survey. *ACM Computing Surveys*, 50(1):2:1–2:45, 2017.

[85] M. Sadoghi, M. Jergler, H.-A. Jacobsen, R. Hull, and R. Vaculín. Safe Distribution and Parallel Execution of Data-Centric Workflows over the Publish/Subscribe Abstraction. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(10):2824–2838, 2015.

[86] P. Salehi, C. Doblander, and H.-A. Jacobsen. Highly-available content-based publish/subscribe via gossiping. In *Proceedings of the 10th International Conference on Distributed Event-Based Systems (DEBS)*, pages 93–104, 2016.

[87] H. Schonenberg, R. Mans, N. Russell, M. Nataliya, and W. M. P. van der Aalst. Process Flexibility: A Survey of Contemporary Approaches. In *CIAO! / EOMAS*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30, 2008.

[88] C. Schuler, H. Schuldt, and H. Schek. Supporting Reliable Transactional Business Processes by Publish/Subscribe Techniques. In *Proceedings of the 2nd International Workshop on Technologies for E-Services (TES)*, pages 118–131, 2001.

[89] Y. Shatsky and E. Gudes. TOPS: a new design for transactions in publish/subscribe middleware. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS)*, pages 201–210, 2008.

[90] A. Slominski, V. Muthusamy, and R. Khalaf. Building a multi-tenant cloud service from legacy code with docker containers. In *2015 IEEE International Conference on Cloud Engineering, IC2E*, pages 394–396, 2015.

[91] J. K. Strosnider, P. Nandi, S. Kumaran, S. P. Ghosh, and A. Arsnajani. Model-driven Synthesis of SOA Solutions. *IBM Systems Journal*, 47(3):415–432, 2008.

[92] Y. Sun, R. Hull, and R. Vaculín. Parallel Processing for Business Artifacts with Declarative Lifecycles. In *Proceedings of Confederated International*

Conferences: CoopIS, DOA-SVI, and ODBASE On the Move to Meaningful Internet Systems (OTM), pages 433–443, 2012.

[93] K. D. Swenson. *Mastering the Unpredictable*. Meghan-Kiffer Press, 2010.

[94] S. Tai, T. A. Mikalsen, I. Rouvellou, and S. M. Sutton. Dependency-Spheres: A Global Transaction Context for Distributed Objects and Messages. In *Proceedings of the 5th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 105–115, 2001.

[95] W. M. P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 2013, 2013.

[96] W. M. P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: a new paradigm for business process support. *Data Knowledge Engineering*, 53(2):129–162, 2005.

[97] L. Vargas, L. I. W. Pesonen, E. Gudes, and J. Bacon. Transactions in Content-Based Publish/Subscribe Middleware. In *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (ICDCS)*, page 68, 2007.

[98] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[99] Wal-Mart Stores Inc. Wal-Mart's Data Synchronization Initiative. `http://www.1worldsync.com/web/us/walmart`, 2009.

[100] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[101] D. Wodtke, J. Weißenfels, G. Weikum, and A. K. Dittrich. The Mentor Project: Steps Toward Enterprise-Wide Workflow Management. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pages 556–565, 1996.

[102] Worflow Management Coalition (WFMC). XML Process Definition Language (XPDL). `http://www.xpdl.org/`, August 2012.

[103] World Wide Web Consortium (W3C). Web Services Choreography Description Language Version 1.0 (WS-CDL). `https://www.w3.org/TR/ws-cdl-10/`, 2005.

[104] S. Yongchareon, C. Liu, and X. Zhao. An Artifact-Centric View-Based Approach to Modeling Inter-organizational Business Processes. In *12th International Conference on Web Information System Engineering (WISE)*, pages 273–281, 2011.

[105] K. Zhang, V. Muthusamy, and H.-A. Jacobsen. Total Order in Content-Based Publish/Subscribe Systems. In *Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS)*, pages 335–344, 2012.