Dynamic Variability Support in Context-aware Workflow-based Systems

Aitor Murguzur Ibarguren

Supervisors:
Dra. Goiuria Sagardui Mendieta
Dr. Salvador Trujillo González

A thesis submitted to Mondragon Unibertsitatea
for the degree of Doctor of Philosophy

Department of Electronics and Computer Science
Mondragon Goi Eskola Politeknikoa
Mondragon Unibertsitatea
Arrasate, May 2015

To my wife, Edurne.

Declaration

Hereby I declare, that this work is my original authorial work, which I have worked out by my own. All sources, figures and literature used during the elaboration of this work are properly cited and listed in complete reference to the due source.

Advisors: Dra. Goiuria Sagardui Mendieta, PhD and Dr. Salvador Trujillo González, PhD.

Abstract

Workflow-based systems are increasingly becoming more complex and dynamic. Besides the large sets of process variants to be managed, process variants need to be context sensitive in order to accommodate new user requirements and intrinsic complexity. This paradigm shift forces us to defer decisions to run time where process variants must be customized and executed based on a recognized context. However, few efforts have been focused on dynamic variability for process families.

This dissertation proposes an approach for variant-rich workflow-based systems that can comprise context data while deferring process configuration to run time. Whereas existing early process variability approaches, like Worklets, VxBPEL, or Provop handle run-time reconfiguration, ours lets us resolve variants at execution time and supports multiple binding required for dynamic environments. Finally, unlike the specialized reconfiguration solutions for some workflow-based systems, our approach allows an automated decision making, enabling different run-time resolution strategies that intermix constraint solving and feature models.

We achieve these results through a simple extension to BPMN that adds primitives for process variability constructs. We show that this is enough to efficiently model process variability while preserving separation of concerns. We implemented our approach in the LateVa framework and evaluated it using both synthetic and real-world scenarios. LateVa achieves a reasonable performance over run-time resolution, which means that can facilitate practical adoption in context-aware and variant-rich workflow-based systems.

Acknowledgements

I don't have words, or the space, to properly thank the people that have influenced this dissertation. Still, I have to try.

First and foremost, I'm extremely grateful to my supervisor Goiuria Sagardui for the support and guidance that she gave me during the thesis period. Goiuria gave me the freedom to do whatever I wanted, at the same time continuing to contribute valuable feedback.

IK4-Ikerlan for being supportive and for offering me the opportunity to complete the thesis. Karmele kept me focused and on-track, and always kept a steady hand needed at the prow. Salva's attention to product-lines is unparalleled and contagious.

And of course, my heartfelt thanks to Xabi, Santi and all the Software Production area members: their help with this thesis in discussing ideas, helping with the paper work, and so forth is beyond my gratitude.

Linh and Schahram for making possible my stay at TU-Vienna. They have taught me much of what I know about writing in science, and improved my thinking about distributed systems and workflows in countless subtle, significant ways. The rest of the team at Distributed Systems group also provided indispensable feedback on an early stage of the thesis. You all made my stay so fruitful and enjoyable!

I'm impossibly grateful to my family and friends. My dear parents Felipe and Maria Luisa, my brother Mikel, Amaia, Anton, Axun, Aitor, Jasone, Liher, and Paul. All of them were there for me throughout the four years of this research. I love you guys!

And finally, to my wife, Edurne. This dissertation, and everything else, is for her.

Contents

List of Figures

List of Tables

Acronyms

AOP Aspect-oriented programming. 36

API Application Programming Interface. 43, 46, 56, 70, 73, 81, 97, 105, 109, 111

BPEL Business Process Execution Language. 13, 32, 33, 35–37, 57

BPM Business Process Management. 12, 14, 24, 25

BPML Business Process Modeling Language. 12, 13, 28, 31

BPMN Business Process Model and Notation. 2, 13, 28, 34, 35, 37, 42, 44, 51, 53, 57, 59, 60, 104

BVR Base-Variation-Resolution. 44, 49, 50, 66, 67

CVL Common Variability Language. 22, 35, 38, 49, 112

DSPL Dynamic Software Product Line. 1, 2, 6, 12, 23, 24, 26, 45, 48, 57, 66, 69, 70, 78, 96, 109

EPC Event-driven Process Chains. 28, 57

FM Feature Model. 19, 20, 22, 36–38, 48, 53, 57, 58, 60, 62, 65, 66, 78, 107

FODA Feature-Oriented Domain Analysis. 22, 57

IRI Internationalized Resource Identifier. 64, 67

KPI Key Performance Indicator. 91

OEM Original Equipment Manufacturer. 90

OWL Web Ontology Language. 37

PLC Programmable Logic Controller. 82, 84, 87, 89, 90, 96

QoR Quality of Result. 75–79, 98

QoRM Quality of Result Model. 76, 77

QoS Quality of Service. 34, 35

RDR Ripple Down Rules. 31

RFID Radio-frequency Identification. 82, 83, 85, 87

SOC Service-oriented Computing. 16, 24

SPL Software Product Line. 1, 12, 20–22, 24–26

SPLE Software Product Line Engineering. 20, 21, 38, 43, 46, 47, 49, 50

TTR Time-to-resolution. 88, 95, 100–103

UDDI Universal Description, Discovery and Integration. 35

WMS Warehouse Management System. 82–85

WSCoL Web Service Constraint Language. 35

WSDL Web Service Description Language. 36

YAWL Yet Another Workflow Language. 31

Chapter 1

Introduction

The past few years have seen a major change in software engineering, as software developers need to reach new requirements and context changes in a flexible manner, as well as reduce design-time and costs while maintaining the quality and delivery of the software. As a result, more and more industries have adopted code reuse as a means to speed up their software development process.

A well-established approach for employing with changing requirements is Software Product Lines (SPLs). An SPL is a set of software products that are closely related (commonality) on a particular market segment, but that exhibit significantly different requirements (variability) [Cle02]. Over the last decade, a wide range of variability-modeling and product line practices have been proposed for maximizing reuse [SR13]. However, SPL solutions still remain challenging in dynamic environments with frequent context changes, which often require run-time adaptation. This has prompted the emergence of Dynamic Software Product Lines (DSPLs) that leverage context awareness and dynamic variability to defer product configuration to run time [Hal08b].

DSPLs have been featured in many different research areas, like in workflow-based systems [SR13], which form the core of this dissertation. In this type of systems, a workflow[1] is a first-class language

[1]We use the terms *workflow* and *process* interchangeably throughout this work.

construct which orchestrates a coordinated set of activities achieve a common goal [Aal04a]. Workflows can share variability characterized by a number of common and variable activities, representing process variants. Process variability deals with a set of similar processes (process variants) and adjust them (process configuration) to meet custom requirements and context changes.

Process variability modeling and conceptualization have been supported by peer-reviewed publications in the literature [Fan12, Döh14, Ayo15]. A major concern for process variability in a DSPL, though, is the context-aware configuration of variants [Rei14]. Here, context information influences process configuration, so the system can dynamically configure variants on the basis of contextual information and effectively deal with dynamic situations at run time. Yet, context-aware dynamic process configuration has found poorly suited, leading to a wide range of reconfiguration approaches [Mur14d].

For example, Worklets [Ada06] enables re-binding of process fragments guided by rules, and Provop [Hal10] enables context-aware process configuration at design-time, as well as reconfiguration. From service-based processes, systems like Discorso [Can08], DyBPEL [Bar12], and MoRE-WS [Alf14] offer similar reconfiguration capabilities, i.e., process variants are configured at build-time and changed during execution. With processes requiring flexibility to be applied in dynamic settings, it seems that workflow-based systems are bound to require dynamic variability for context-aware and variant-rich processes.

This dissertation argues that, instead, we can provide a holistic solution that not only captures process variability, but enables context-aware process configuration at run time. In particular, we show that a simple extension to Business Process Model and Notation (BPMN), which is interpreted by the LateVa (Late Variability for Context-Aware Smart Workflows) engine, can enhance flexibility. The resulting framework has the following characteristics:

1. It supports separation of concerns by representing process variability in separated models (base model, fragment, and variability model), enabling reuse during process variability modeling, and offering easier development of process families.

2. It enables an automated run-time selection of fragments based on contextual information. Indeed, our approach yields process variants using multiple binding and subsequently resolves dynamic variation points at run time.

3. It achieves reasonable performance at fragment binding (around 14 ms). Indeed, it is able to select suitable fragments for variation points with significantly less computation time than the status quo, e.g., around 300 ms for process model and instance migration in [Bar12] and for model operations in [Alf14].

4. It is highly amenable for large-scale process families, allowing to bind alternative variants at run time.

We implemented our approach in the LateVa framework, including LateVa modeler as the process variability foundation, and LateVa engine for run-time variability processing. We evaluated our framework using both real user applications and traditional benchmarks for quantitative analysis. Our implementation provides reasonable performance and scalability at fragment binding, as well as the first platform to let developers compose and execute context-aware process variants at run time. However, some limitations remain in the evaluation such as a limited support for context data types and the use of real-life workflows.

Thesis statement: *A process variability model based on contextual information can efficiently support process configuration at run time.*

In the remainder of this chapter, we explain some of the motivations for context-aware dynamic configuration of process variants and highlight the main contributions.

1.1 Problem Statement

Today's workflow-based systems need to tailor reuse practices, as well as offer greater flexibility to deal with large number of variants, context changes, and intrinsic complexity. While approaches like C-EPC [Ros07] and Worklets [Ada06] aimed to capture fairly process variability and adaptation, researchers and practitioners have developed more and more specialized systems for new application domains. Recent examples include Provop [Hal10], Template & Rules [Kum12] and vBPMN [Döh13] for business processes, VxBPEL [Kon09], DyBPEL [Bar12] and MoRE-WS [Alf14] for service-based processes, and others.

Although such specialized workflow systems seem like a natural way to scope down the challenging problems in the dynamic environment such as run-time adaptation, they also come with several drawbacks:

1. **Lack of reuse assistance:** Many variability-aware workflow systems need to solve the same underlying problems, such as process reuse. For example, a process developer would naturally try to reuse process fragments from similar process structures. Having similar process variants, many approaches aggregate all variants in a configurable process; however, reuse need to be addressed, separating commonality and variability description, in order to generalize it as the preferred work practice.

2. **Context awareness:** In the presence of context information, process configuration should be context-aware to be able to customize variants based on fluctuating context resources.

However, in reality, the efforts have been limited and thus context-aware process configuration still remains challenging.

3. **Run-time decision making:** Run-time decision making is just in its infancy partially because most process variability approaches assume that process variant customization at design-time is sufficient. In such systems, adaptation and reconfiguration are commonly adopted which involve a more intrusive supervision and enforcement compared to run-time variability.

4. **Scalability concerns:** Large-scale process families require an efficient engine to configure variants in a reasonable amount of time. Even for run-time approaches, scalability must be folded to achieve a better performance.

Because of these limitations, a unified dynamic variability approach for workflow systems would have significant benefits in terms of flexibility, especially for context-aware and variant-rich settings.

1.2 Objectives and Contributions

To address this problem, we introduce a new framework, called LateVa - *Late Variability for Context-aware Smart Workflows*, that allows for configuring process variants at run time. The insight behind LateVa is that although various approaches support schemes and process flexibility mechanisms (e.g., adaptation, reconfiguration), they all lack *an efficient context-aware run-time variability for process variants*. With run-time variability capabilities and multiple binding support, all the decisions for variant configuration can be deferred to execution phase, capturing the context data in each placeholder activity resolution. LateVa offers such functionality for variant-rich workflow-based systems, in a manner that is efficient and automated.

In particular, we use LateVa to model and execute variability. In the modeling phase, LateVa follows the "design by reuse" approach using

fragments as first-class entities, but at run time, it adds new properties, such as context awareness, multiple binding support and automated decision making, that current approaches lack (see Chapter 3). We discuss the main contributions below.

Variability and Context Management. This first contribution enables process variability and context modeling in detached models. By working on process variability with a variability model, we aim at defining a process family and at identifying commonalities and variabilities along process variants. This model also permits to define constraints among variants and context related features. Such context information is represented in a context model, which ultimately determines the context data needed. Also, at the context gathering process, this model is used to determine which context features in the variability model are mapped to the context variables retrieved from underlying data services.

Run-time Variability and Multiple Binding. As the second contribution, the LateVa engine defers process configuration to run time. It exposes base models as services, which can be invoked and started for process variant execution and configuration. Since variant configuration is context sensitive, the engine is capable of collecting, interpreting and applying contextual information to automatically decide which of the available fragment options are most appropriate. It also supports multiple binding to change configuration behavior, i.e., the engine can resolve variability using both startup time and pure run time binding times for DSPLs.

This exploration aside, we also show empirically that we can implement some of the specialized workflow-based systems in use today, as well as large-scale process variants, using LateVa.

1.3 Publications

Contributions entail publications. The presented dissertation lead to the following publications, grouped by topic and chronologically according to the year in which they were published:

On approaches for enabling process flexibility:

- [Mur13a]: Flexible Processes and Process Mining: A Brief Survey. IBM Technical Report (2013)
- [Mur14d]: Process Flexibility in Service Orchestration: A Systematic Literature Survey. International Journal of Cooperative Information Systems. (2014)

On the framework for context-aware dynamic configuration of process variants:

- [Mur13b]: Process Variability through Automated Late Selection of Fragments. In: VarIS workshop, CAiSE. (2013)
- [Mur14b]: Context-aware Staged Configuration of Process Variants@Runtime. In: International Conference on Advanced Information Systems Engineering (CAiSE). (2014)
- [Mur15a]: Dynamic Variability Support in Workflow-based Systems: An Evaluation of the LateVa Framework. In: ACM/SIGAPP Symposium On Applied Computing (SAC). (2015)
- [Mur15b]: Runtime Variability for Context-aware Smart Workflows. IEEE Software. (2015)

On the multi-perspectives support for process variability:

- [Mur13c]: Multi-perspective Process Variability: A Case for Smart Green Buildings. In: IEEE International Conference on Service Oriented Computing and Applications (SOCA). (2013)

- [Mur14c]: On the Support of Multi-perspective Process Models Variability for Smart Environments. In: International Conference on Model-Driven Engineering and Software Development (MODELSWARD). (2014)

On context variability for context-aware dynamic configuration:

- [Mur14a]: Context Variability Modeling for Runtime Configuration of Service-based Dynamic Software Product Lines. In: DSPL workshop, SPLC. (2014)

On the abstractions for process selection and configuration:

- [Mur14e]: DRain: An Engine for Quality-of-Result driven Process-based Data Analytics. In: International Conference on Business Process Management (BPM). (2014)

Other publications:

- [Mur12b]: Towards a Model-based Hybrid Service Composition for Dynamic Environments . In: European Conference on Service Computing and Cloud Computing (ECSOC PhD Symposium). (2012)

- [Mur12a]: Model-driven and Planning for Service Composition in Dynamic Heterogeneous Environments. In: 6th Advanced School on Service Oriented Computing (SummerSOC Poster and PhD session). (2012)

1.4 Research Methodology

This dissertation has been accomplished by following the guidelines for performing research in information systems as described by Vaishnavi and Kuechler[2]. This research methodology consists of five main steps:

1. **Awareness of the problem**: The awareness of the problem provides the opportunity to unlock new findings. In our case, we identified the problem to resolve and stated it clearly (Chapter 1).

2. **Suggestion**: The second step includes a comprehensive state-of-the-art to extract potential open research issues and thus suggests a solution to the problem stated based of the formulated statement. The improvements of that solution need to be compared with already existing solutions on the field. We considered the following outputs here: a state-of-the-art review and a tentative design of the solution (Chapters 2, 3, and 4).

3. **Solution design and development**: Tentative design is further developed and improved in this phase. The novelty is primarily in the design but can be applied to implementation. Specifically, we designed the main building blocks of LateVa and implemented associated modules (Chapters 5 and 6).

4. **Evaluation**: The evaluation step validates the solution in real and synthetic scenarios to analyze the performance, feasibility, and scalability of the approach. In our case, we tested LateVa using two examples and a quantitative analysis (Chapter 7).

5. **Conclusion**: Finally, this phase analyzes the results of the specific research effort in order to obtain conclusions, limitations, and delimitate areas for further research (Chapter 8).

[2] `http://bit.ly/1xewPx3`

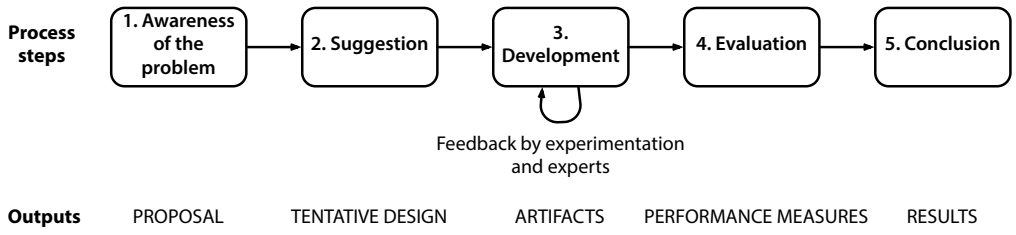


Figure 1.1: Research methodology.

1.5 Outline

This dissertation is organized as follows: Chapters 2 and 3 position the research by reviewing current literature on process variability approaches. Chapter 4 introduces the LateVa framework and its main building blocks to enable context-aware configuration of process variants at run time. Chapter 5 covers the modeling phase for implementing different models required on LateVa. Chapter 6 leverages these models and introduces the LateVa engine to support context awareness and dynamic configuration at run time. Chapter 7 discusses the applicability of LateVa using experimental scenarios, as well as its limitations and possible extensions. Finally, we conclude and discuss potential areas for future work in Chapter 8.

Part I

State of the Art

"*When you're young, you're not afraid of what comes next. You're excited by it.*"
-Dave Grohl

Chapter 2

Background and Context

This chapter introduces the fundamental background necessary to understand the dissertation as a whole. A more specific attention concerns the notion of process variability and DSPLs (dynamic variability), which are explored in detail in Chapter 3.

Structure of the chapter. This chapter first introduces the notion of Business Process Management (BPM) (Section 2.1), including process life-cycle, workflow-based systems, and process flexibility aspects with a special emphasis on process variability. Then, SPLs and DSPLs are characterized (Section 2.2). Finally, the thesis context is exposed (Section 2.3), providing some insights on the motivation and objectives which led to the development of this work.

2.1 Business Process Management

BPM is concerned with the supervisory of business processes including concepts, methods, and techniques to support the design, configuration, enactment, and evaluation within the BPM life-cycle [Wes12, Dum13]. A business process is described by a *process model* (or schema) to define how information (control-flow) and activities are passed from one participant to another using a Business Process Modeling Language (BPML). The Workflow Management Coalition

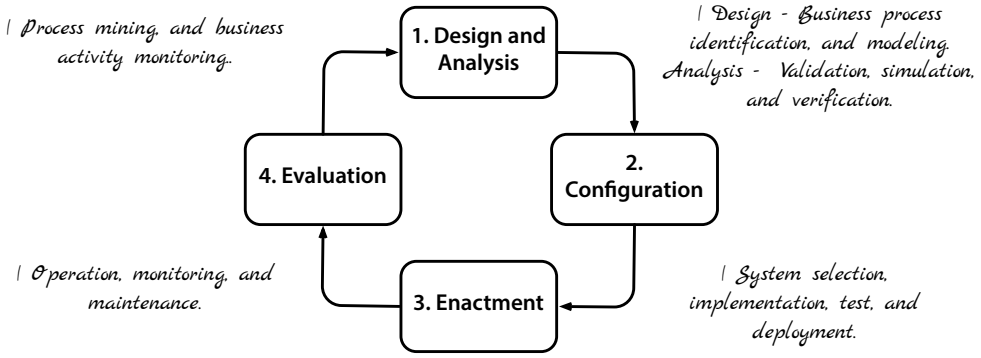


Figure 2.1: Process life-cycle adopted from [Wes12].

(WfMC)[1] defines a *process* as *"a formalized view of a business process, represented as a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal"*. Such processes are regularly handled through the completion of repetitive and routine activities.

2.1.1 Process Life-cycle

The process life-cycle is a continuous loop consisting of four phases [Wes12], see Figure 2.1. The life-cycle begins with *process design and analysis*, where processes are identified, validated, simulated, verified, and finally represented as process models. During the *configuration* stage, process models are developed using a BPML [Kop08, Mil10, Pic12] (e.g., BPMN, Business Process Execution Language (BPEL), Petri Nets, UML Activity Diagrams or Event-driven Process Chains (EPCs)), and tested to ensure their correctness prior to deployment. The aim of the *enactment* phase is to execute such models within a workflow execution engine according to the instructions issued by the implemented process schemes. The process monitoring task leverages

[1] `http://www.wfmc.org`

information about the current execution of process instances in order to evaluate them. The results from the *evaluation* phase are used to close the process life-cycle and continuously improve processes, e.g., processes can be evaluated through process mining [Gun08].

2.1.2 Workflow-based Systems

A workflow-based system is the major paradigm in use in today's BPM practice [Aal13].[2] Concretely, a workflow-based system refers to a software system that helps to define, execute, and monitor a given sequence of tasks, arranged as workflows [Aal04b]. Those workflows (or pipelines) are normally well-structured, involving software systems and people, and executed in a controlled environment where their structure and behavior is not regularly updated. However, in recent years, increased process flexibility has been a primary concern for the implementation of workflow-based systems [Web09].

Process flexibility

Process flexibility has been researched for many years [Aal13]. The term *process flexibility* has been defined in various ways by different authors [Sad01, Reg05, Reg06b, Reg07, Pes08]. In this dissertation, we adopt the definition proposed by Sadiq et al. [Sad01] to consider *process flexibility* as *"the ability of the workflow process to execute on the basis of a loosely, or partially specified model, where the full specification of the model is made at run time, and may be unique to each instance"*.

Starting from these definitions, a number of taxonomies are also present in the literature to categorize different flexibility needs [Hei99, Reg06a, Sch08, Bal10, Rei12]. For instance, Schonenberg et al. [Sch08] revisited the concept of flexibility by Heinl et al. [Hei99] and extended the original taxonomy. The authors proposed a taxonomy description of four distinct approaches that can be taken to enhance flexibility of a

[2] A workflow is a part of BPM; conversely BPM is a superset of workflow.

Table 2.1: Comparison of process flexibility taxonomies.

Heinl et al. [Hei99]	**Schonenberg et al. [Sch08]**	**Reichert and Weber [Rei12]**
Flexibility by Selection	Flexibility by Design	Variability
–	Flexibility by Deviation	Planned Adaptation
Instance Adaptation	Flexibility by (Momentary) Change	Unplanned Adaptation
Type Adaptation	Flexibility by (Evolutionary) Change	Evolution
Late Modeling	Flexibility by Underspecification	Looseness

particular process. If we compare the taxonomy of Schonenberg et al. vs. Heinl et al., we can conclude that one flexibility type is abandoned and another is added, as shown in Table 2.1.

Reichert and Weber [Rei12] proposed an analogous classification to define flexible processes according to four major flexibility needs, namely: variability, adaptation, evolution, and looseness. This taxonomy adopts comparable terms to those described in Schonenberg et al.'s taxonomy as follows (see Table 2.1):

- **Variability**: handles different process variants (similar processes) depending on the particular process context. Process variability is further discussed in Section 2.1.3.
- **Adaptation**: represents the ability to deal with changes and consequently adapt process behavior and its structure at run time. Two types of drivers may trigger process adaptation:
  - *Exceptions*: planned adaptations or exception handling mechanisms that are pursued depending on the type of the foreseen exception detected during process execution. Exception handling patterns can be found at [Rus06].
  - *Special situations*: unplanned adaption mechanisms that require structural adaptations of a single process instance. Such structural changes can be carried out by *change patterns* (i.e., high-level change operations, e.g., inserting a process

fragment between two edges), or *change primitives* (e.g., add node, add edge, etc.). Change patterns are discussed at [Web08].

- **Evolution**: represents the ability of process instances to change as the corresponding process schema evolves. Here, the assumption is that a process model will affect new and running instances, migrating only running compliant instances to the new specification, such as in ADEPT2 [Rei09]. More patterns and examples are thoroughly discussed in [Rei12, Sba14].

- **Looseness**: relates to knowledge-intensive processes which keep parts of the process unspecified during build-time to deal with, for instance, unpredictability, non-repeatability, and emergence [Kem11]. This means that future course of action depends on knowledge acquired through each activity execution. To achieve such loosely-specified models four different decision deferral patterns are distinguished:

  - *Late selection*: the selection of the actual content for the placeholder activity is deferred to run time. This pattern can be considered as a synonym for "late binding" or "dynamic binding" from the Service-oriented Computing (SOC) such as in CEVICHE [Her10]. More approaches have been evaluated in [Mur14d].
  - *Late modeling*: the modeling of the placeholder content is deferred to run time. Therefore, it goes one step beyond than late selection, due to the run-time modeling nature. Pockets of Flexibility approach [Sad01] fits into this category.
  - *Late composition*: it defers workflow creation to run time. Over time the initial plan is refined by incorporating new knowledge from the execution phase. Summaries of current approaches can be found at [Mur14d].

– *Ad-hoc composition*: this method enables on-the-fly composition of process fragments from the process repository, evolving instances incrementally during run time by executing activities in an ad-hoc manner. Examples are given in [Rei14].

In the following, we further explain the details of process variability, a topic which plays a significant role in this dissertation.

2.1.3 Process Variability

Process variability deals with *process variants* that share common parts of a core process (base model) whereas concrete parts change from variant to variant. For instance, as shown in Figure 2.2, four process variants conform to the same *process family*, where $\{A, C, D\}$ activities are common to all process variants $\{P_{v1}, P_{v2}, P_{v3}, P_{v4}\}$. Each variant is valid on a particular context, i.e., the *configuration* of a process variant depends on contextual information or user requirements.

A configurable process model or a base model is customized for each *process context*, where a configuration can be selected from a list of available choices (fragments), after which it is no longer possible to change. This selection is called *binding*, and the stage in the life-cycle at which binding occurs is called *binding time*, including configuration time, deployment time, startup time, and pure run time [Cap14a]. The latter is homonymous with the "late selection" pattern.

Overall, considering the process life-cycle of Section 2.1.1, four major requirements for process variability can be derived from [Rei14]:

- **Modeling**: "Design by reuse" should be promoted to create new consistent variants by taking over existing variants. To correctly represent variability, three main aspects need to be determined [Tor12]: (i) what parts of a process model may change, (ii) what alternatives exist in/for each variation point, and (iii)

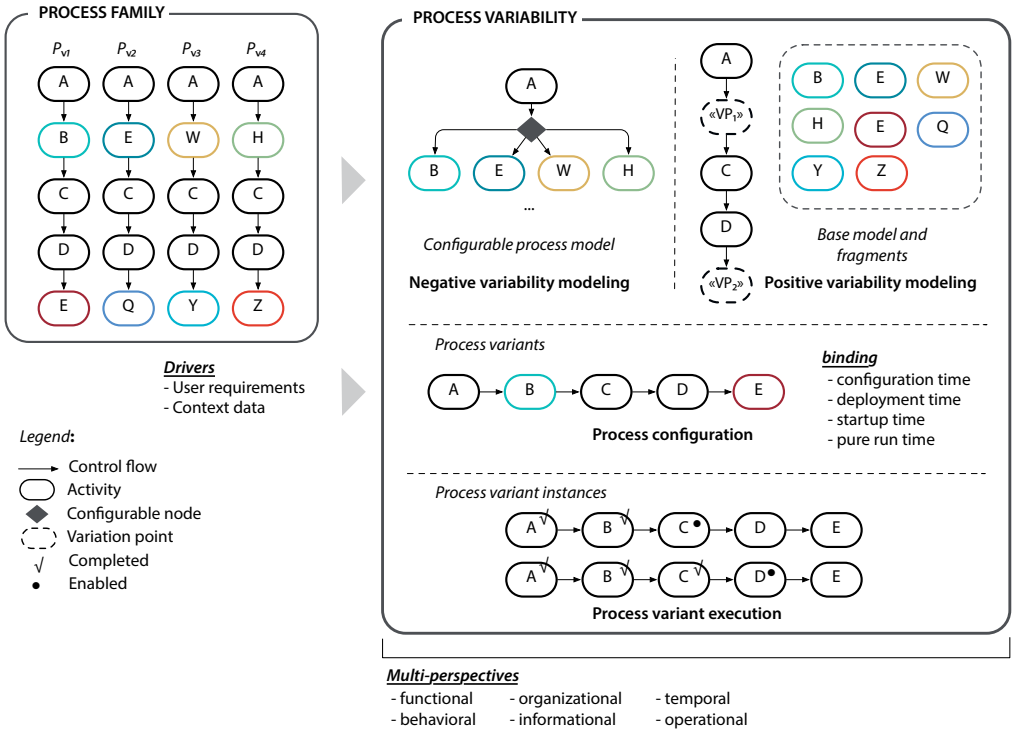


Figure 2.2: Process variability.

conditions that make these alternatives being selected. For that purpose, two principal modeling bents have been raised (see Figure 2.2): (i) *negative variability* (also referred to as variability by restriction [LR13], behavioral variability [Tor12], or single-element based variability [Döh14]) where a *configurable process model* gathers all process variants in a single reference process model, and (ii) *positive variability* (or variability by extension, structural variability, or fragment-based reuse approach) where process variability is modeled using separated models such as *base models* for identifying commonalities and *fragment* for individualities. Both methods allow for removing redundancies by representing variants' commonalities only once; however,

the latter manages variability independently clearly reducing its impact over commonalities, as well as the complexity of managing large sets of process variants.

- **Variant configuration**: The process variant configuration may involve different drivers for customization (e.g., contextual information, user requirements, etc.). Such configuration can be captured by using different *decision support techniques* such as Feature Models (FM), decision tables, and questionnaire models [LR13]. The first two techniques provide abstractions to allow the system and/or user-based reasoning in terms of domain concepts rather the process modeling elements. The latter offers guidance to guide users in making the right decision, for instance in terms of recommendations. Despite the technique used, context-aware automated process configuration is enforced in dynamic settings where process context is only available at run time [Rei14].
- **Execution**: During execution, the workflow-based system should allow for dynamic variability by deferring variant binding (includes startup time and pure run time bindings, see Figure 2.2).
- **Maintenance and optimization**: All reconfiguration and run-time changes should be traced to ensure their correct application and enforcement to running instances.

As shown in Figure 2.2, process variability can be realized from different process variability perspectives. In [Ayo15], the authors examined process variability from five different process perspectives:

- **Functional perspective**: takes activities as first class entities (atomic or complex), so variability is centered on how activities change from variant to variant.
- **Behavioral perspective**: focuses on control-flow variability, i.e., how different control connectors (e.g., gateways) and control edges (e.g., arrows) can change among variants.

- **Organizational perspective**: represents the different stakeholders or actors that are in charge of executing process variants.
- **Informational perspective**: covers data flow variability, i.e., process variability is described based on how data and data flow are passed during execution.
- **Temporal perspective**: describes the temporal constraints that may restrict the scheduling and execution of activities (e.g., the time of an activity to start or finish, etc.).
- **Operational perspective**: focuses on the implementation of atomic process activities and deals with variability of different services that can be bound to a particular activity.

In this dissertation, we take the functional perspective and analyze how FMs from Software Product Line Engineering (SPLE) can be employed as a decision support technique to defer process variability to run time.

2.2 Software Product Lines

In software development, developing from scratch is a tedious and costly task. Instead, software reuse is adopted in industries to realize improvements in time to market, cost, and quality, among others.

SPLE is a software development approach that employs reusable software units (also referred to as *core assets*) for creating software [Cle02]. A SPL refers to a set of products that are closely related (commonality) but exhibit differences in their requirements (variability). Concretely, the main characteristic of a SPL is the shift from a single system point-of-view to a set of related software products where commonality and variability are exploited.

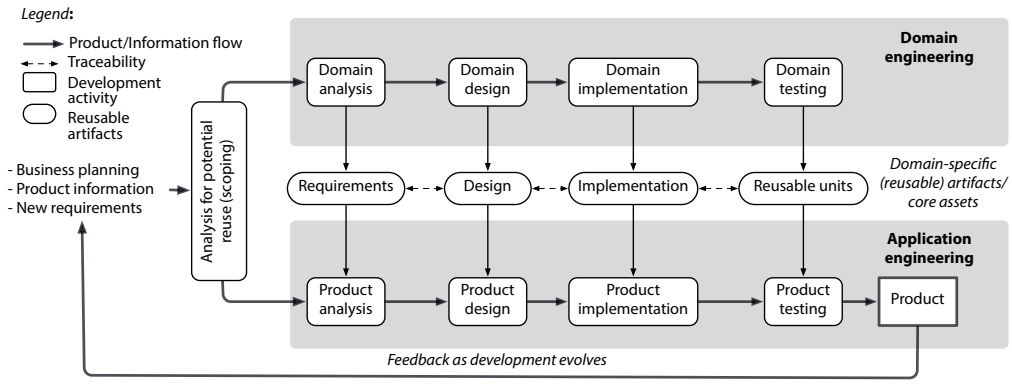


Figure 2.3: SPL dual life-cycle adopted from [Hal08b].

2.2.1 SPL Dual Life-cycle and Variability Management

One of the SPLE principles is its dual life-cycle, proposed by Hallsteinsen et al. [Hal08b]. As shown in Figure 2.3, this life-cycle consists of two principal phases:

- **Domain engineering**: is the responsible for the development of reusable core assets, i.e., the architecture of a SPL is built here.
- **Application engineering**: customizes core assets for developing new products according to application specific requirements, i.e., product realization is achieved here.

Another key aspect of the SPLE is the variability management. Variability management deals with software models across different stages of the product life-cycle in order to produce a family of configurations [Sin07]. Such variability is commonly described in terms of *variation points* and *variants*. The former identifies software parts that contain variability and in which variant binding occurs. The latter, in turn, refers to particular instances of realizing variability.

Variability modeling is intended to capture the essence of how one product is similar, but still different from another. It allows to specify

common and variable parts of products using a variability modeling language [Cza12, Ber12]. In essence, there are two principal ways to represent variability [Met14]: *integrated variability* - where dedicated or specialized variability constructs (e.g., stereotypes) are introduced into a existing modeling language, and *orthogonal variability* - where variability is represented in a dedicated and separated model.

For integrated variability, since the introduction of Feature-Oriented Domain Analysis (FODA) [Kan90], over 40 different FM dialects have been proposed [Ben10], grouped into three main categories: (i) *basic feature models* offering mandatory, alternative and "or" features, as well as "requires" and "excludes" cross-tree constraints, (ii) *cardinality-based feature models* offering UML-lie multiplicities for features [*m..n*], and (iii) *extended feature models* to add arbitrary feature attributes (e.g., integer values) [Kar14]. Orthogonal variability also comprises some reference languages such as the OMG standard proposal called Common Variability Language (CVL) [Hau12].

SPLs have been successfully adopted to improve the process of software design, development, and maintenance, developing a wide range of product lines in a number of different domains such as embedded systems [Bos12] and service-oriented systems [Cha07a, Lee12]. Relevant findings on variability modeling adoption on industries are reported in [Ber13, Vil14]. However, in traditional SPLs, once the product is created, it turns difficult to make changes on it without stopping the system. Consequently, the execution must be interrupted in order to provide the system with new features and/or different behavior (e.g., adding or removing variants), thereby creating a new product, while it could be quite similar to the previous one, results into a completely different product.

2.2.2 Dynamic Software Product Lines

Product line approaches remain challenging in dynamic environments with frequent changes such as context-aware or ubiquitous systems,

often requiring run-time adaptation. These new requirements have prompted the emergence of DSPL [Hal08b,Cet08,Mor08] that leverage context awareness and *dynamic variability* (also referred to as *run-time variability* [Cap11]) to defer product configuration to run time.

In a DSPL, a configurable product is capable of dynamically (re-)bind variation points at run time by considering context information. Once changes are detected (e.g., depending on external sensors or new user requirements) the system makes decision on which features of the configurable product must be activated or deactivated, and executed the decision by using run-time binding. Capilla et al. [Cap14a] define the following requirements for DSPL realization:

1. **Run-time variability support and management**: a DSPL must support run-time configuration of products by activating/deactivating features, as well changes in the structural variability by reconfiguration mechanisms (e.g., adding/removing features) [Bar15].

2. **Multiple and dynamic binding**: a DSPL may change from context to context, so the system should adapt accordingly providing multiple binding support. This will allow for staged-configuration [Cza04] where choices are resolved in different binding times, e.g., using different variability transformation strategies [Cet09b].

3. **Context-awareness and self-adaptation for autonomic behavior**: contextual information should be exploited to dynamically adapt variants and/or select new alternatives autonomously based on the context conditions.

Starting from 2008, DSPLs have served as baseline for different research initiatives [Mor09, Par09, Cet10]. However, to the best of our knowledge, limited research has evaluated the usage of DSPLs for process families.

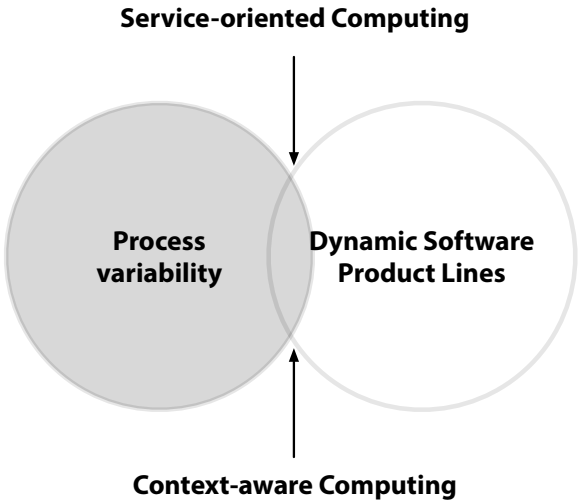


Figure 2.4: Thesis context - Venn diagram.

2.3 Thesis Context

The association between BPM and SPLs has been documented in [SR13, Val13a] (see Table 2.2). Bearing in mind such mapping, this dissertation combines aspects from process variability and DSPLs to enable context-aware configuration of process variants at run time. Apart from process variability and DSPLs, we also adopt some notions from context-aware computing and from SOC (see Figure 2.4). The former enables the annotation of process variants with contextual information and the latter exposes core processes and fragments as services, while enabling the system to actively influence process configuration.

Table 2.2: Knowledge mapping between BPM and SPLs concerning variability. Adapted from [Val13a].

BPM Concept	SPL Concept	SPL Explanation
Configurable (business) process modeling language	Variability modeling language	Variability modeling language allows describing commonality and variability in SPL products.
Configurable (business) process model	Product family architecture	A product family architecture is the unification of all systems architectures in a product family, capturing the architectural commonalities and variabilities in a product family and guiding developers when new application instances are derived.
Configuration decision	Configuration decision	Decision when binding a variation point.
Core (business) process	Core asset	A core asset is reusable software artefact across at least two SPL instances. It can be as coarse-grained as the product family architecture or as fine-grained as small components, e.g., classes.
(Business) process configuration	Product instantiation	Product instantiation means creating a specific software product using a SPL. It removes from the SPL architecture all unneeded functionality and therefore selects pre-implemented variants.
(Business) process correctness	Well-formed SPLs	Well-formed SPLs are obtained when a set of constraints are captured during the configuration of software artefacts, determining if a configuration is valid.
(Business) process line or process family	Product family	A product family captures commonalities between software products for the product family, promoting reuse of core components across related software products.
(Business) process perspective	SPL perspective	A perspective aggregates multiple views to refine the variant space of the SPL.
(Business) process flexibility	Flexibility	Flexibility is a key concern is SPL development. Core assets must easily change to accommodate the requirements of different products.
(Business) process instance	SPL instance	An instance is a running product of the SPL.
(Business) process reference model	Reference architecture	Setting up and describing a reference architecture for a product line provides a basis for member instantiation.
(Business) process similarity	Commonality	A SPL captures commonalities between software products of a product family.
(Business) process type	Artefact type	Artefacts comply with well-formed rules: types.
(Business) process variability	Variability	For a SPL, variability describes the characteristics that vary from application to application within the product family.
(Business) process variant	Variant	A variant in SPL stands for a software component, being design alternatives to resolve the variability in a product family.
Variability mechanism	Variability mechanism	A variability mechanism is a variability realization technique for a product line.
Variation point	Variation point	Variation points are available for systems/users to build variants in a SPL.
Design/build-time and runtime	Binding time and binding mode (static, dynamic)	Development stage when variability is bound.

Chapter 3

State of the Art

The term DSPL was introduced in 2008 by Hallsteinsein et al. [Hal08b] to expose run-time variability and self-adaptive properties for traditional SPLs. Since that date, DSPLs have been applied in various domains such as mobile systems [Gin10], embedded systems [Bos12], and service-based systems [Lee12].

In this dissertation, we examine the convergence between DSPLs and process variability to enable a seamless management of process variants at run time. To this end, the following chapter positions the research by reviewing the existing relevant literature in variability-aware workflow-based systems, with a greater emphasis on approaches supporting run-time variability. This chapter presents and expands upon work published in [Mur13a, Mur14d].

Structure of the chapter. This chapter begins with presenting the existing surveys on design-time process variability (Section 3.1.1). These surveys merely report the existing support for process variability modeling, and thus serve as a means to motivate the need for dynamic variability management. Although some of the criteria of these surveys can be used for run-time analysis, we present additional criteria (Section 3.1.2) for comparing approaches supporting dynamic variability, and provide a summary of the state-of-the-art (Section 3.2).

3.1 Variability on Workflow-based Systems: A Reality Check

In this section, we survey different studies that compare and analyze process variability approaches. We extract meaningful conclusions and insights from the comparative studies, and at the same time collate primary approaches dealing with run-time variability.

Table 3.1: Surveys on design-time process variability.

Year	Paper	Title
2005	Recker et al. [Rec05]	On the User Perception of Configurable Reference Process Models
2008	Nurcan [Nur08]	A Survey on the Flexibility Requirements Related to Business Processes and Modeling Artifacts
2010	Kapuruge et al. [Kap10]	Support for Business Process Flexibility in Service Compositions: An Evaluative Survey
2010	Burkhart et al. [Bur10]	Flexible Business Processes-Evaluation of Current Approaches
2010	Aiello et al. [Aie10]	Requirements and Tools for Variability Management
2012	Dijkmana et al. [Dij12]	Managing Large Collections of Business Process Models - Current Techniques and Challenges
2012	Fancinato et al. [Fan12]	A Survey on Reuse in the Business Process Management Domain
2012	Torres et al. [Tor12]	A Qualitative Comparison of Approaches Supporting Business Process Variability
2012	Ayora et al. [Ayo12b]	Towards Run-Time Flexibility for Process Families: Open Issues and Research Challenges
2013	Grambow et al. [Gra13]	Challenges of Applying Adaptive Processes to Enable Variability in Sustainability Data Collection
2013	Valença et al [Val13b]	A Systematic Mapping Study on Business Process Variability
2013	La Rosa et al. [LR13]	Business Process Variability Modeling: A Survey
2014	Döhring et al. [Döh14]	Configuration vs. Adaptation for Business Process Variant Maintenance: An Empirical Study
2014	Reichert et al. [Rei14]	Lifecycle Management of Business Process Variants
2014	Mechrez et al. [Mec14]	Modeling Design-Time Variability in Business Processes: Existing Support and Deficiencies
2015	Ayora et al. [Ayo15]	VIVACE: A Framework for the Systematic Evaluation of Variability Support in Process-Aware Information Systems

3.1.1 Design-time Variability Surveys

Process variability approaches have been studied for decades, and are now experiencing a resurgence of interest. For instance, the VIVACE evaluation framework [Ayo15] has been proposed for comparing process variability approaches. Some of these approaches have been analyzed in different surveys (see Table 3.1) in an attempt to compare the existing support for reuse and design-time process variability. We have analyzed the results and drawn conclusions as to what has worked and what can be improved. The findings are summarized below.

Process variability modeling. Process variability modeling started to impact in 2005, with a major rise in recent years [Dij12]. For instance, in 2005, the PESOA project [Puh05] was carried out to propose an approach for the development and customization of process-oriented software families and transferred to several languages (e.g., UML Activity Diagrams, BPMN, UML State Machines, and Matlab/Simulink). Popular BPML includes Event-driven Process Chains (EPCs) and BPMN [Val13b]; however, although some extensions have been proposed in the literature (e.g., C-EPC [Ros07], C-YAWL [Got08], vBPMN [Döh11], VxBPEL [Kon09]) there is still not a standard extension for process variability [Ayo15]. Similarly, [Ayo15] does not show a clear evidence in favor of a process variability technique (negative or positive variability modeling). In [LR13], only 5 out of 19 approaches support positive variability, so there is space to balance. The authors also state that positive variability gives more freedom to developers so they can add or modify parts of the model. From cognitive psychology perspective, positive variability might also impose a lower mental effort [Tor12]. In [Döh14], for each of the performance measures of success rate, user contentment and execution speed, vBPMN (positive variability) performs significantly better than C-YAWL (negative variability). However, correctness can be limited by using positive variability modeling [LR13]. Overall, we conclude

that additional research is needed to determine how each variability modeling technique works in different settings, e.g., with large-scale process variants.

Multi-perspectives process variability. Most of the process variability approaches have been focused on control-flow perspective, i.e., they bring functional and/or behavioral perspectives into reality [Mec14]. The same evidence is reported by [Ayo15], where temporal and operational perspectives are largely ignored by most approaches. Although some efforts have been made to integrate multiple perspectives in a coherent and consistent framework [Mee11, Mee12, LR11, Sai14], there is still a room for improvement [Rei14, LR13].

Context awareness. Context awareness is one of the major concerns for context-aware process configuration [Rei14]. However, only a few approaches, such as Provop [Hal08a, Hal10], support contextual information during process configuration. This is specially important not only for design-time process variability, but also for run-time customization [Gra13]. Beyond context information, other approaches drive process configuration based on high-level abstractions such as requirements [Lap07] or questionnaire models [LR07].

Executable process variants. As it was mentioned, several approaches have been proposed to enable process variability modeling [Asa14]. However, a few become executable in practice [LR13]. This means that most of them stay on conceptual modeling and do not glean executable process variants [Rec05, Ayo15]. Other flexibility properties, such as adaptation and late binding, have also been considered by different workflow-based systems [Nur08, Kap10, Bur10].

Run-time variability and reconfiguration. Dynamic environments require more flexibility and competitiveness requires quick development [Fan12]. As highlighted in [Ayo12b], *"current variability proposals do not provide proper support for run-time configuration of process variants."* This means that none of the approaches support the resolution of variation points at run time. In contrast, reconfiguration of process variants has been supported, e.g., by Worklets [Ada06] and Provop [Hal10], but a holistic approach for process flexibility in process families is still missing [Ayo12b]. This point is also emphasized in the findings of [Gra13, Rei14].

Tool-support and validation. Existing tools have been developed to validate the feasibility of the proposed ideas, and thus constitute proof-of-concept prototypes to foster industrial uptake [Ayo15, LR13]. While many of such tools have been evaluated through case studies or experiments, there have been no quantitative evaluations to the present day. This indicates that more empirical research is required to evaluate primary studies supporting process configuration [Tor12].

3.1.2 Run-time Variability Approaches

In the following, we describe the related work on run-time process variability. The approaches are presented chronologically according to the year in which they were proposed.

Run-time variability criteria

Table 3.2 details the new criteria for approaches supporting dynamic process variability. Each approach is classified according to two main criteria: (i) modeling aspects, and (ii) execution environment. We also include for convenience some general properties such as evaluation support, tool support, and type of scenario.

Table 3.2: Comparison criteria for run-time variability approaches.

Code	Criteria	Definition
Modeling aspects		
M1	Process variability modeling technique	Describes if positive or negative variability is used.
M2	BPML	Indicates the language to represent the process models.
M3	Variability modeling language	If any, indicates the language to represent the variability models.
M4	Context modeling	If case of context-awareness, determines how context information is modeled.
M5	Reuse	If reuse practices are integrated and supported in the process variability modeling process.
Execution environment		
E1	Run-time configuration (late selection)	Defines the capability of the workflow-based system to configure and execute specific regions of a process variant at run time, i.e., late binding.
E2	Run-time reconfiguration (adaptation)	Specifies if the approach allows to switch from the current process variant instance to another one.
E3	Multiple binding	Concretes if multiple bindings are supported.
E4	Context awareness	Determines if context information is used for late binding during execution.
E5	Degree of automation	Defines the degree of automation in variability resolution time, e.g., automated, system-supported or manual.
E6	Decision-making	Concretizes the way process configuration is performed (e.g., rule-based, goal-based, constraint-based, etc.), i.e., *-driven configuration.
E7	Evolution of the process family	Determines if the system may propagate the changes of a configurable process model to already running process variant instances.
General properties		
G1	Empirical evaluation	Indicates if the system scales up and down, be relevant to process families of any size, including context data, and provides an empirical evaluation.
G2	Tool-support	Details if the approach provides tool support for managing process variants.
G3	Scenario	Specifies the scenario or application domain in which run-time variability is applied.

Adams et al. [Ada06] present the Worklets approach to build and execute loosely-specified process models by applying re-binding.

- *Modeling*: *Worklets* has been implemented as a Yet Another Workflow Language (YAWL). Variants' restrictions are defined by a set of rules, namely Ripple Down Rules (RDR). During process specification, each activity can be associated with a set of sub-process fragments, which can be dynamically extended at run time. The *Worklet specs* repertoire is used to store generated process models, RDR, and Logs.

- *Execution*: At run time, choices are made dynamically from the sub-process fragments repertoire by considering data attributes and values associated with a specific instance, using a pre-defined set of rules. Hence, when activities become enabled, the adaption can be done by RDRs. Depending on the given rules, the YAWL process activity is replaced by the selected fragment and executed by the YAWL engine. Users may also adjust fragment selection

by rejecting the proposed fragment or adding new rules. In [Ada07] authors extend the Worklet service with an exception handling sub-service called Exlets, which provides both expected and unexpected process exception handling.

Charfi et al. [Cha07b, Cha09] present a plug-in architecture to provide self-adaptation support for BPEL processes by means of monitoring and adapting aspects, i.e., using an aspect-oriented extension to BPEL, namely Ao4BPEL.

- *Modeling*: Every BPEL process activity is an extension point where the plug-in can execute adaptation logic. Inside plug-ins, two types of aspects can be used: (i) *monitoring aspects*, which collect information to decide whether adaptation is needed, and (ii) *adaptation aspects*, which handle events detected by the monitoring aspects.

- *Execution*: Once the BPEL processes are deployed, the engine detects quickly fault handlers that can be invoked to identify failures in invocations including pre-examination, post-examination, and wrapping. These advices invoke the adaptation aspects (plug-ins) which replace the faulty partner service. Such plug-ins can be added or removed from the proposed architecture during composition execution (user-based provisioning). Hence, it clearly benefits extensibility and enables support for service adaptation by means of generic requirements (service unavailability, partner service non-functional property changes through WS-Policy, and non-functional requirements through Service Level Agreements (SLAs)) and application specific adaptation support.

Karastoyanova et al. [Kar09] present the BPEL'n'Aspects approach by combining the standard BPEL, the publish/subscribe paradigm, and

WS-Policy to improve the flexibility of service orchestrations.

- *Modeling*: BPEL processes can be mapped to aspects in three different forms: (i) for all process instances, (ii) only for a specific instance, and (iii) for a subset of instances. Aspects are represented using WS-Policies and WS-Policy Attachments as a means for associating aspects with BPEL processes in a non-intrusive manner.

- *Execution*: During execution, the BPEL engine is able to signal events (by WS-Notification) to the broker, which manages the subscriptions of the aspects/Web services, and coordinates the adaptation of process logic. This mechanism allows attaching aspects to processes at run time without changing the process schema itself.

Koning et al. [Kon09] present the VxBPEL approach to address dynamic composition of Web services by providing variability constructs in the language level and treats changes as first-class entities.

- *Modeling*: VxBPEL allows to capture variation points, variants, and realization relations among variation points, in order to deal with variability of service-based processes (described in BPEL). Concretely, it provides three types of variability support: *service replacement* to change a service by one with the same/different interface, *changing the parameters* of a service, and *changing the composition* of the system. The latter defines service fragments that can be modeled in a variation point as variants.

- *Execution*: VxBPEL is executed by the ActiveBPEL engine. Once the BPEL process is deployed the ActiveBPEL is able to customize suitable variants based on fixed configuration information. Such data structure associated with configurable variation points can be changed at run time through JMX, enabling reconfiguration.

Hallerback et al. [Hal10] describe the Provop approach for modeling and managing large collections of process variants by means of base models and adjustment points (variation points).

- *Modeling*: In the modeling phase, a base model can be defined in BPMN using adjustment points (variation points). These points define the regions of a base model to which adaptation can be applied at run time. Four change operations are defined: INSERT fragment, DELETE fragment, MOVE fragment, and MODIFY attribute. Change operations can be grouped in options to structure change patterns. Some of the options can also be correlated using different kind of constraints, such as implication, mutual exclusion, application order, hierarchy, and at-most-n-out-of-m-options. During configuration, the user selects a sequence of options and applies them to a base model, configuring a process variant. Such configuration can also consider contextual information [Hal08a].

- *Execution*: Once the process variant is created, the process variant is executed according to pre-established fragments. However, Provop allows for process variant changes during execution, i.e., run-time reconfiguration of process variants, to adequate process execution to contextual changes.

Ardagna et al. [Ard11] propose the Discorso (Distributed Information Systems for Coordinated Service-oriented Interoperability) framework to offer a service-based solution for modeling and managing business processes.

- *Modeling*: At design-time the Discorso framework lets designers to define business processes in BPMN, abstract services, fragments, and Quality of Service (QoS) constraints. Starting from the extended BPMN specification, it derives a skeleton

of a WebML hypertext extended with primitives. It also translates the fragments of the BPMN process that correspond to automated activities into BPEL. An extended Universal Description, Discovery and Integration (UDDI) registry stores concrete Web services and associates them with QoS profiles.

- *Execution*: At run time, WebRatio orchestrates the whole process and invokes concrete Web services by implementing a late binding mechanism through wrappers. This means that if a concrete Web service is faulty or violates a constraint (these expressions are specified in a variant of Web Service Constraint Language (WSCoL)), the framework finds a suitable substitute satisfying the QoS constraints. Hence, the system monitors constraints and in the event of QoS violations, triggers the adaptation of the running process instance.

Baresi et al. [Bar12] present the DyBPEL approach based on CVL and BPEL for managing process variability and reconfiguration.

- *Modeling*: The interplay between CVL and BPEL is conducted by various elements: (i) *CVL choices* which represent an user-centric description of variability, (ii) a *base model* representing BPEL processes that include process variables, activities, and partner links, (iii) a *CVL library* which exemplifies the additional BPEL process elements employed to generate variants, (iv) *substitutions* which are related to each variant, and (v) *production realization* function which allows to select variants to being included in the process configuration.

- *Execution*: Once the process variants is generated, the *Coordinator* deploys and stores the process schema in the repository. At run time, the *Coordinator* is the responsible of detecting process models changes and deriving those

modifications to the *Runtime Modifier*. The *Runtime Modifier* intercepts the execution of the running process instances and applies changes in the repository. This is achieved through Aspect-oriented Programming (AOP). Finally, the *BPEL Modifier* handles the migration of process definitions to the new specification. More recently, the same author has proposed a solution for the run-time evolution of service-based processes [Bar14].

Cubo et al. [Cub11, Cub13] present an extension of the DAMASCo framework, the so-called FM-DAMASCo, to deal with service variability expressed in featured models for dynamic service composition.

- *Modeling*: During the modeling phase, the user can define context information in XML files, service signatures in Web Service Description Language (WSDL), business processes in BPEL, and service variability in a FM representing the service family. Then, Context-Aware Symbolic Transition Systems (CA-STSs) are extracted from the BPEL services, which implement the client(s) and services.

- *Execution*: In each service request, the DAMASCo model transformation (CA-STS) and the semantic-based service discovery tasks are launched. Once the suitable service is encountered, the self-adaptation process is executed. This way the approach is capable of adapting the service flow during execution time. Recently, co-authors follow a similar path for managing transactional service variability [Gam15].

Alferez et al. [Alf14] propose the MoRE-WS solution based on a semantically rich variability model to support the dynamic adaptation of service compositions and context awareness.

Table 3.3: Summary of modeling and general properties support for run-time variability.

Approach	M1	M2	M3	M4	M5	G1	G2	G3
Worklets - [Ada06]	Positive	YAWL	RDR	rule-based	✓	-	✓	casualty treatment
AO4BPEL - [Cha07b, Cha09]	n/a	BPEL	-	-	-	-	✓	travel domain
BPEL'nAspects - [Kar09]	n/a	BPEL	-	-	-	-	✓	not specified
VxBPEL - [Kon09]	Negative	BPEL	VxBPEL	not specified	-	-	✓	loan approval
Provop - [Hal10]	Positive	BPMN	Options	OWL	✓	-	✓	vehicle repair
Discorso - [Ard11]	n/a	BPMN/BPEL	-	-	-	-	✓	order processing
DyBPEL - [Bar12]	Positive	BPEL	CVL	-	✓	✓	✓	smart homes
DAMASCo - [Cub11, Cub13]	Positive	BPEL	Feature models	CA-STSs	✓	✓	✓	navigation service
MoRE-WS - [Alf14]	Positive	BPEL	Feature models	OWL	✓	✓	✓	online book shopping
LateVa - [Mur14b]	Positive	BPMN	Feature models	OWL	✓	✓	✓	automated warehouses

- *Modeling*: In order to support dynamic adaptation, the framework models abstractions that represent context information (a *context model* in Web Ontology Language (OWL)), dynamic configurations of the service compositions (a *variability model* using a FM), the composition itself (a *composition model* in BPMN), and the adaptation policies necessary to adapt existing service compositions at run time (a *weaving model*).

- *Execution*: Model-based reconfiguration is implemented on top of the MoRE-WS engine, an extension of MoRE [Cet09a]. The framework oversees context changes and periodically updates the context model according to the information collected by the SALMon context monitor. This altered model is used as a baseline for adaptation reasoning (SPARQL) and model changes (EMFMQ). Finally, changes are materialized by mapping BPMN variants to BPEL fragments as reported in [Ayo12a].

Summary on Run-time Variability Approaches

Tables 3.3 and 3.4 summarize existing support for process variability at run time. We use a check mark (✓) if the approach proposes or deals with the different criteria, and a dash (-) in the opposite case.

Regarding modeling and general properties, positive process

Table 3.4: Summary of execution properties support for run-time variability.

Approach	E1	E2	E3	E4	E5	E6	E7
Worklets - [Ada06]	-	✓	-	✓	system-supported	rule-based	-
AO4BPEL - [Cha07b, Cha09]	-	✓	-	✓	automated	aspect-based and SLAs	-
BPEL'n'Aspects - [Kar09]	-	✓	-	✓	automated	aspect-based and WS-Notification	-
VxBPEL - [Kon09]	-	✓	-	-	automated	data structured based and exposing functionalities through JMX	-
Provop - [Hal10]	-	✓	-	✓	manual	rule-based	-
Discorso - [Ard11]	✓	-	-	-	automated	QoS-based service discovery	-
DyBPEL - [Bar12]	-	-	-	✓	automated	aspect-based, interrupting each activity execution	✓
DAMASCo - [Cub11, Cub13]	-	✓	-	✓	automated	semantic service matching	-
MoRE-WS - [Alf14]	-	-	-	✓	automated	constraint-based	✓
LateVa - [Mur14b]	✓	-	✓	✓	automated	constraint-based	-

variability modeling is often adopted for representing run-time variability. Such variability can be represented in several ways, but variability modeling techniques from SPLE such as CVL and FMs are likely common, as reported in [Mon08]. Context awareness is only addressed by a few approaches. Similarly, only 3 out of 9 show an empirical evaluation.

As for run-time properties, we find marginal support for run-time configuration of process variants. Most of the approaches are usually based on reconfiguration or adaptation mechanisms to modify process instances or service compositions at run time. Some of the approaches use contextual information and event rules to trigger adaptations. However, such approaches do not offer support for run-time configuration of process variants, neither for multiple binding.

3.2 Concluding Remarks

In this chapter we have surveyed primary studies in the literature that are closely related to the main contribution of this dissertation. Firstly, we have introduced 16 surveys that analyze different approaches dealing with reuse and design-time process variability. Then, we have reviewed and compared relevant approaches (9 in total) primarily focused on run-time flexibility (variability, reconfiguration, and

evolution), with the aim of strengthen the main goals of LateVa.

This chapter concludes the second part of this document dedicated to the study and analysis of the background and state-of-the-art. The next three chapters describe the contributions of this dissertation. We start with a global overview of the main building blocks of LateVa (Chapter 4), to present the modeling (Chapter 5) and run-time resolution details (Chapter 6) in subsequent chapters.

Part II

Contribution

"Failure is an option here. If things are not failing, you are not innovating enough."
-Elon Musk

Chapter 4

Foundations for Context-aware Configuration of Process Variants at Run time

In workflow-based systems, process models need to provide means to modify their structure and behavior dynamically in order to cope with context changes and run-time exceptions. In recent years, this has been tackled by process flexibility approaches [Web09, Rei12], including variability, adaptation, evolution and looseness (see Chapter 3). However, few efforts have been focused on enabling context-aware process configuration at run time. Concretely, dynamic variability makes use of late selection pattern to defer process configuration , i.e., it combines variability management and late binding. This is specially challenging for context-aware and variant-rich scenarios where process context may influence the customization of each individual variant. This issue has been the focus of our approach and thus constitutes the main contribution of this dissertation. We have proposed a framework, called LateVa, to enable context-aware dynamic process configuration.

Structure of the chapter. This chapter first introduces the main building blocks of LateVa (Section 4.1). Next, we describe the process of domain engineering (Section 4.2), making special emphasis on the design phase artifacts, and focus on the run-time phase in application engineering (Section 4.3). Finally, we end with a summary of LateVa in relation to both design and run-time phases (Section 4.4).

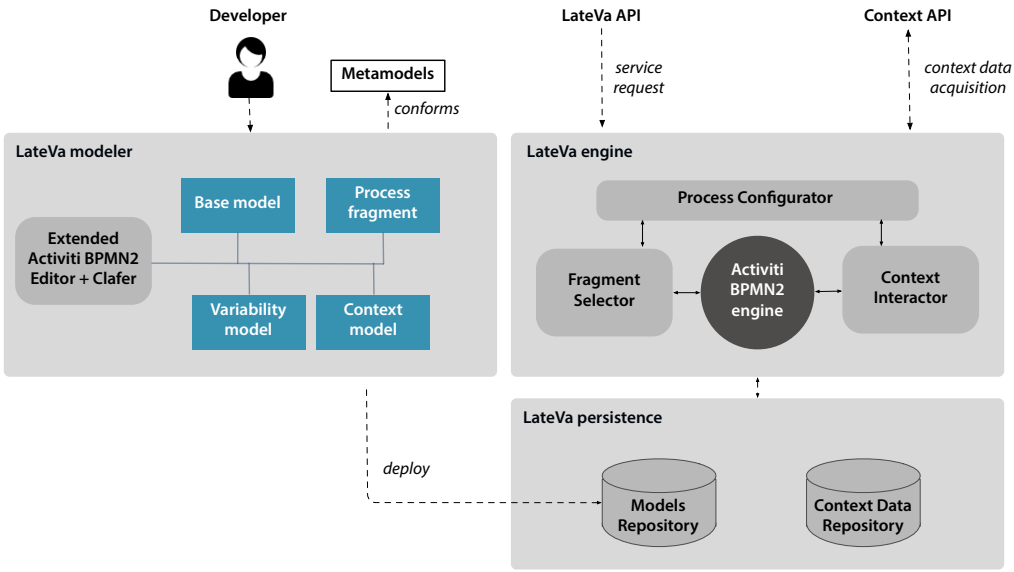


Figure 4.1: LateVa overview.

4.1 LateVa: A Global Overview

The LateVa framework allows for modeling, configuration, processing, and execution of process variants based on context information (the main building blocks are illustrated in Figure 4.1). In order to isolate process variants, the developer may sketch or reuse *base models* (common activities) and *fragments* (variants) (expressed as BPMN 2.0 diagrams), and employ a *variability model* and a *context model* for variability/domain specification. Base models may hold custom decision points, namely *variation points*, to make reactive fragment decisions based on contextual information. Once developed, all these models, except the context model, are deployed to the ***Models Repository*** and made immediately available to the engine.

In each service request, the ***Process Configurator*** searches for available base models that can accomplish the request, and make use of the process engine to start base model execution. During the execution,

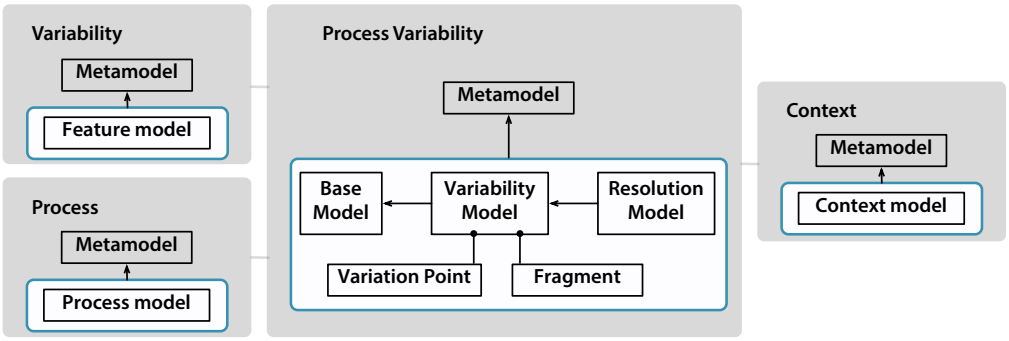


Figure 4.2: A fresh look on domain engineering.

a base model instance might need to interact with different services exposed in the context through the ***Context Interactor***. Data retrieved from the Context Application Programming Interface (API) is saved in the ***Context Data Repository*** and utilized by the ***Fragment Selector*** to make an automated decision on candidate fragment using a variability model and a solver.

These steps entail the two phases of a SPLE: the *domain engineering* phase in which the core assets (base models and fragments) are created and the *application engineering* phase where individual models (variability models and context models) are defined and resolved either in the startup time or dynamically at run time.

4.2 Domain Engineering

In this section, we describe the domain engineering process as illustrated in Figure 4.2. This process aims to develop the core assets (base models and fragments) that will be leveraged latter in the application engineering phase. Assets are divided into two categories: *metamodels* - abstract assets that are used to create concrete assets, and *models* - models derived from metamodels.

Process variability as core metamodel. This metamodel aims to embrace the commonality and variability of a process family. The ultimate goal is to provide a language that allows developers to describe process variability. This is achieved by adopting the BVR (Base, Variation, Resolution) approach [Bay06] to model commonality, variability and relationships in separated models.

Context awareness for augmenting process variability. This metamodel allows to extend process variability metamodel with contextual information. Such context metamodel details different context types and context data that can be mapped to a variability specification (i.e., context features), and then used by the engine to obtain contextual information, including explicit support for on-demand computation of context-specific data.

Metamodels for variability and process modeling. We use feature modeling (i.e., Clafer[1]) for variability description and BPMN 2.0 as a target process modeling language (i.e., Activiti[2]). The motivation behind these choices relates to the fact that both implementations are open source and commonly adopted [Ber13], written in Java, and provide extension points for customized behavior.

From these metamodels, concrete models are obtained to reflect commonality and variability of all process family members. Such concrete models refer to *base models* (expressing what is the same among processes) and *fragments* (representing process individualities) that are written by hand in the domain engineering phase and reused in the application engineering phase.

[1] `http://clafer.org`
[2] `http://activiti.org`

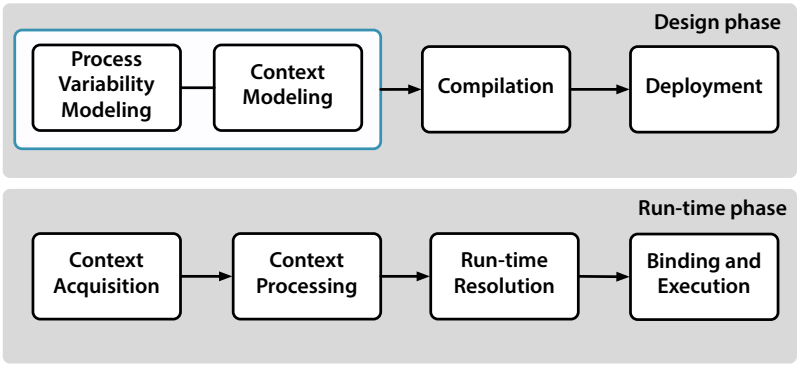


Figure 4.3: A fresh look on application engineering.

4.3 Application Engineering

The application engineering phase concerns the development of individual assets. Those assets represent the whole process family and restrictions, including all variants that can be configured at run time.

4.3.1 Design Phase: Variability Modeling and Deployment

The design phase involves building the skeleton of the process family. In this phase, we can determine two models such as a *variability model* and a *context model*. The former describes the process variability spectrum and limits the configuration of each process variant to context information. The latter serves as common domain description necessary to represent contextual information.

This phase comprises four main steps (see Figure 4.3), including: *process variability modeling* where core assets (base models and fragments) and a variability model are used to define the whole process family, *context modeling* to relate context data schema to variability description, and the *compilation and deployment* phases.

4.3.2 Run-time Phase: Variability Configuration and Enactment

The run-time phase aims to configure a process variant based on contextual information, as in a DSPL. One advantage of using DSPLs is that product derivation (process configuration in our case) is deferred to run time. Hence, the developer can have different binding times to automatically infer decision points based on contextual clues.

The run-time phase consists of four main steps (see the lower panel of Figure 4.3), including: *context acquisition* to collect context data from context services/APIs, *context processing* for mapping context data values to variability description, *run-time resolution* to decide on a suitable fragment based on current context data for a particular variation point, and fragment *binding and execution*. These steps enable the dynamic process configuration on LateVa.

4.4 Concluding Remarks

In this chapter we have presented the fundamental building blocks of LateVa for enabling context-aware dynamic configuration of process variants. We emphasized on the two main phases of SPLE, and introduced the types of models that we employ to model and execute context-aware process configuration. Details of each of these two phases are covered in consecutive chapters.

Chapter 5

Design Phase: Process Variability and Context Modeling

Two of the most important challenges in the convergence of SPLE and workflow-based systems concern to process variability modeling and dynamic process configuration [SR13]. The former refers to how to describe, manage, and implement the commonality and variability of an existing process family (a set of related process variants), whilst the latter deals with a smart workflow derivation at run time.

This chapter focuses on the former to cover the *process variability modeling and deployment* phases at design time, with a special emphasis on context and process variability modeling. We start by introducing the process variability and context metamodels, and provide hints on how feature and process metamodels are mapped to process variability constructs. Then, we describe the mandatory steps for the modeling, compilation, and deployment phases.

Structure of the chapter. This chapter first presents the challenges for the design phase (Section 5.1). We then outline the main building blocks of the design phase by means of the so-called metamodels (Section 5.2), and describe the steps to derivate a process family (Section 5.3). Finally, we revisit the challenges for the design phase and conclude the chapter (Section 5.4). This chapter presents and expands upon work published in [Mur13b, Mur13c, Mur14c].

5.1 Design Phase Challenges

In Chapter 1, we have identified two main challenges for the design of workflow-based DSPLs, such as "design by reuse" and context awareness. In the following points, we present a more detailed definition of those challenges in order to explicitly deal with the problems related to separation of concerns, process variability and contextual information:

1. **Ensure clean separation of concerns**: to correctly represent process variability two principal modeling bents have been raised: *negative variability* - where a configurable process model gathers all variants in a single reference model, and *positive variability* - where "design by reuse" is promoted by separating process variability into disjointed models. Both remove redundancies by representing variants' commonalities only once; however, only the latter allows for *separation of concerns* by modeling commonality and variability in detached models.

2. **Promote reuse**: during the process variability modeling phase, it is possible that the same fragments can be utilized by two or more process variants, or even shared among process families. Such fragments can be understood as a connected and reusable process structures, which can give an easier and faster development of variant-rich workflow-based systems. Therefore, the *fragment repository* should serve as the main repository to store, manage, and access reusable fragment libraries [Sch11, Eka11].

3. **Mapping between features, context, and process variability artifacts**: FMs enable the clear specification of software variability as a feature tree, hierarchical format [Ach12]. The *mapping* that holds between the features in a FM, context data, and inherent process variability constructs can be complex to define. This is especially the case in the presence of context features (features

mapped to context data) and process variability related features (variation points and fragments), which may further complicate the interactions between features.

4. **Provide context variability support**: although the mapping between features and context data can be realized by an appropriate context mapping mechanism, we also need to deal with the *data ambiguity* problem. Different context data may belong to the same feature, i.e., context models may differ syntactically and semantically from each other. Such data heterogeneity may cause unexpected behavior in data processing and mapping, allowing identical context values to be sent within different context keys.

To face these challenges, in the following sections, we present the main building blocks of LateVa backed by the domain engineering and application engineering phases from SPLE.

5.2 Design Phase: Domain Engineering

This section describes in detail the metamodels that have been implemented in LateVa for enabling context-aware process variability modeling. We differentiate four metamodels, but particularly focus of the former two as part of the contribution. For the latter two, we briefly arrange them and present external references for further reading.

5.2.1 Process Variability Metamodel

In our case, we have defined a process variability metamodel inspired from the BVR approach (named from Base, Variation, Resolution models) presented at [Bay06], and recently revisited at [Hau14]. BVR is a language built on CVL [Hau12], and enhanced in the VARIES

Figure 5.1: The BVR approach by Bayer et al. [Bay06].

project[1]. Basically, the BVR approach states the separation of models' commonality, variability and possible configurations into isolated artifacts such as *base models*, *variation models* and *resolution models*, as illustrated in Figure 5.1. The major advantage of using BVR though, is that there can be more than one variation model for each base model. Thus, developers can associate as many as variability models to a single base model, representing variability scenarios at different levels.

We have adopted the same high-level concepts and relationships to define variability, and expand them to incorporate new process-related constructs. Nevertheless, we rename the term "variation model" into "variability model", due to its greater adoption in the SPLE community [Hau08]. Our process variability metamodel is shown in Figure 5.2. It aims to support the functional perspective introduced in Chapter 2 to represent process' entities and related variability.[2]

Base Model

The `BaseModel` represents the commonality shared by a process family (a set of similar process variants) in a particular domain. This model can be seen as the intersection or Greatest Common Denominator (GCD) of all considered variants, comprising the shared behavior of all of them. Such variants share a common part of a core process (captured in the base model) whereas concrete parts (expressed as fragments) fluctuate from variant to variant. In LateVa, a base model is built by

[1] `http://www.varies.eu/`

[2] Although not covered in this dissertation, the author has also worked on a modeling and execution alternative to show that multi-perspectives process variability can be realized [Mur13c, Mur14c].

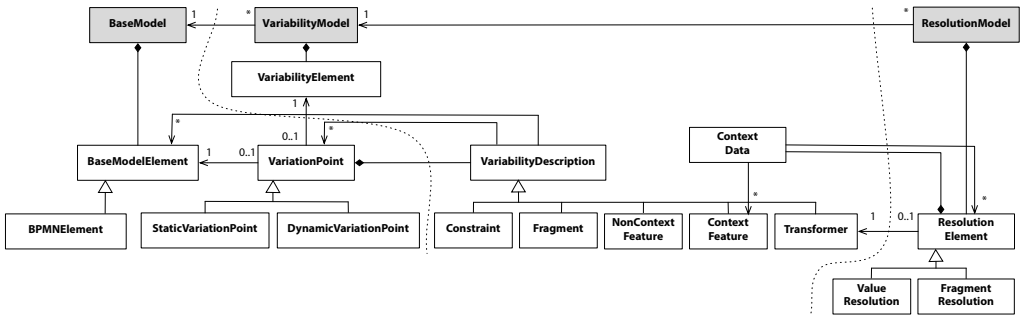


Figure 5.2: Process variability metamodel.

an extended BPMN2 process modeling language.

A `BaseModelElement` stands for any kind of BPMN2 element in a base model. The relationship between base model element and variation point has zero-to-one cardinality since not all base model elements can be affected with variability (see Figure 5.2). Apart from BPMN2 elements, a base model may also hold placeholder activities (variation points) to point out those parts of the base model where different alternatives can be applied depending on a specific context. Formally, a base model is defined as follows:

Definition 1 (Base Model)**.** *A base Model $\mathcal{BM}$ is defined as a 2-tuple <$\mathcal{C}$, $\mathcal{VP}$>, being $\mathcal{C}$ the set of commonalities of a process family and $\mathcal{VP}$ the set of feasible variation points. Every base model instance $\mathcal{BMI}$ derived from $\mathcal{BM}$ consists of a set of commonalities and a set of variation points:* $\forall\ \mathcal{BMI} \in \mathcal{BM}, \mathcal{BMI} = (\{\mathcal{C}\}, \{\mathcal{VP}\})$.

In short, a `VariationPoint` identifies the part in a base model where variant binding occurs. It represents a configurable base model element in which an alternative (expressed as fragment) can be selected based on context data, after which it is no longer possible to change. In our own case, the use and placement of variation points emphasizes two principal factors: points out a local or remote data endpoint from

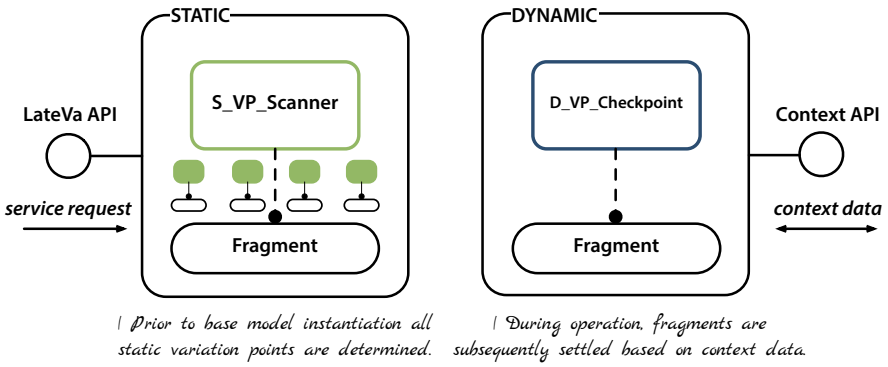


Figure 5.3: Variation points behavior.

which context data is gathered, and indicates a placeholder activity in a base model where variability may take place.

The behavior of a variation point becomes context dependant, requiring itself to flock explicit context data for final fragment resolution (dynamic variability). However, not all variation points might be subjected to dynamic context. For instance, in non-critical scenarios, i.e., where the resolution is not sought under the run-time configuration, a limited amount of variation points can be pre-wired before base model instantiation (static variability), and thus do not require any context interaction at all. To cover such behaviors, we distinguish two types of variation points:

- **Static:** a `StaticVariationPoint` denotes a static placement within a base model. This will be fully determined at built-time by means of selecting a suitable fragment alternative from a set of a limited number of options (see Figure 5.3).
- **Dynamic:** a `DynamicVariationPoint` determines a dynamic placement within a base model that will be bound at run time to an applicable fragment based on context data (see Figure 5.3).

Variability Model

A `VariabilityModel` assembles all the particularities introduced by each process variant. Accordingly, it offers abstraction for the base model and its variation points when making a customization, in addition to decision support. Such variability can be expressed using variability modeling techniques. In LateVa, we make use of FMs[3]. This allows us to make variability elements explicit by describing variability in a hierarchical, tree format, and left-over cross-tree constraints. A variability model is formalized as follows:

Definition 2 (Variability Model)**.** *A variability model $\mathcal{VM}$ is defined as a 3-tuple <$\mathcal{VE}$, $\mathcal{F}$, $\mathcal{O}$>, being $\mathcal{VE}$ the variability sub-model made up of the set of domain non-context features, context-features and variation points, $\mathcal{F}$ the set of possible applicable fragments for each variation point, and $\mathcal{O}$ the constraint sub-model made up of restrictions.*

As depicted by Figure 5.2, a `VariabilityElement` refers to any feature, attribute or constraint element in a FM. A FM stands for a family of features in the domain and relationships among them. A `VariabilityDescription` represents the actual variability of a variation point which is directly related to a variability element. Here, we distinguish five meta-classes:

- **Non-context feature:** a `NonContextFeature` describes any domain feature not related to context data. As in the domain feature modeling, a non-context feature can be of type Mandatory, Optional, Or and Alternative (xor).
- **Context feature:** a `ContextFeature` refers to a variable defined by surrounding context (e.g., fdc1010 : integer). This type of feature can be a boolean condition of type $\{0,1\}$, or an attribute of a primitive data type (e.g., integer). This mapping allows to infer

[3] As found on [Ber13], FMs have gained widespread adoption in industry.

and interpret contextual information from different perspectives [Jar10], e.g., from process variability perspective in LateVa.

- **Constraint:** a `Constraint` represents an optimum condition for variant resolution, expressed as cross-tree restrictions between variability model elements (e.g., A implies B, C excludes D). Such cross-tree relations can be defined at three levels: complex non-context feature/non-context feature, non-context feature/context feature, and context feature/context feature.
- **Fragment:** a process fragment, or simply `Fragment`, describes a single variant realization option for each variation point within a particular base model. This can be related to one or more variation points. Hence, each variation point may have a number of possible implementations (fragments), intended as a reusable granule for process variability.
- **Transformer:** a `Transformer` includes concrete transformations associated to a variability specification. When context values are bound to a particular transformer, it executes the corresponding resolution. In other words, it is the responsible for deciding which features should be activated by the current context data values, i.e., value or fragment resolution.

Resolution Model

A `ResolutionModel` defines resolutions of a variability model. It represents a binding of variability specifications, which can be used to derive a new, specific model. From the process variability perspective, it specifies which of the fragment alternatives described in a variability model are valid for placed variation points considering current context conditions. A resolution model is formalized as follows:

Definition 3 (Resolution Model)**.** *A resolution Model $\mathcal{RM}$ is defined as a 4-tuple <$\mathcal{CV}$, $\mathcal{V}$, $\mathcal{VP}$, $\mathcal{RF}$>, being $\mathcal{CV}$ the set of context variables, $\mathcal{V}$ the*

set of context values, $\mathcal{VP}$ the referred variation points, and $\mathcal{RF}$ the set of relevant fragments.

A set of resolution elements define how a variability model is bound. Concretely, as stated in [Bay06], a `ResolutionElement` represents *"a binding of a variability specification, i.e., it represents a binding of variability"*. This is either a complete binding in which all variability is resolved, or a partial one in which some variability is still present (see Figure 5.2). A resolution has a number of effects which represent the correlation a resolution has on the variability model, such as narrowing a constraint or removing parts of the model. It could either resolve all the variation points at once or subsequently configure each variation point. We distinguish two subtypes:

- **Value Resolution:** a `ValueResolution` determines a value for the variability model where context values are mapped to context features. Hence, LateVa correlates context features within a variability model to those context key/value pairs retrieved from context data.
- **Fragment Resolution:** a `FragmentResolution` represents resolutions for variation points in the variability specification, as well as the corresponding mappings in a base model.

There can be several resolution models pointing to the same variability model. For one resolution model, each transformer can be linked to zero or one resolution element, so not all transformers must be associated to resolution elements. Even in a static variation point description, some variability can be beyond the scope due to higher-level resolutions. In contrast, dynamic variation points are settled by parsing `ContextData` and mapping context value pairs to the corresponding context features.

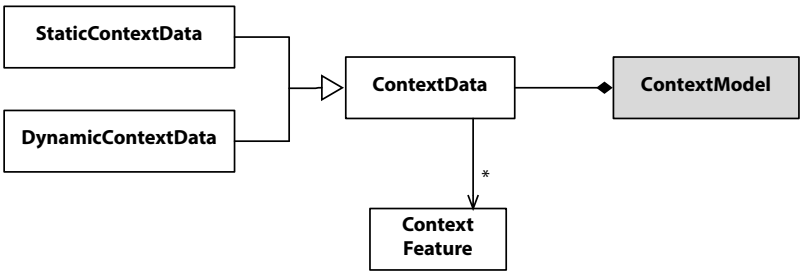


Figure 5.4: Context metamodel.

5.2.2 Context Metamodel

Process variability metamodel allows to model variants of a particular process model, whereby each variant is valid in a specific scenario or context. Each context requirements build the *process context* [Hal08a] and drive the configuration of a particular process variant. *Context* itself can be referred to as a *"any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves"* [Abo99]. Beyond the established definitions, in our case, context describes the context data coming from underlying data sources (sensors, services, APIs) that may directly influence process context, and thus drive dynamic process configuration. All context core concepts set up the context model, enabling contextual knowledge modeling, sharing and reuse [Str04, Bet10].

As illustrated in Figure 5.4, a `ContextModel` describes the data that can be processed in a particular context, including different entity types and their relationships. In LateVa, such `ContextData` is exploited to correlate expected context features in a variability model to those being experienced. Two different context data types are considered:

- **Static Context**: `StaticContextData` represents static preferences of a particular context that are known prior to

base model instantiation. This type of context rarely change over time, so can be foreseen at built-time.

- **Dynamic Context**: `DynamicContextData` describes dynamic context data that change frequently over time and thus can be only accessible at run time. This context type is available when the selection of a dynamic variation point occurs.

Context modeling not only allows for context awareness but also fills the gap between process configuration and process context.

5.2.3 Process Metamodel

Process logic is represented in form of process models using a process modeling language (e.g., BPMN, Petri Nets, UML Activity Diagrams, EPC, or BPEL [Lu07, Gro14]), which stands for a particular process type, consisting of a number of activities to be executed.

As it was stated before, we have adopted the BPMN2 metamodel defined by Object Management Group (OMG)[4] as the de-facto process modeling language in LateVa. Figure 5.5 presents the main concepts behind the specification. In order to integrate process variability constructs as part of the BPMN2 description, we have extended and enriched variation points with `ServiceTask` behavior.

5.2.4 Feature Model Metamodel

With the increasing popularity of DSPLs [Cap14a], several extensions have been proposed in order to enhance FMs (i.e., FODA models or feature diagrams) and add explicit support for cardinalities, feature groups, feature relationships, and collaborative aspects [Cap13, Cap15]. However, context awareness is the key enabler of context variability [Har08, Cap14b], where feature models can be used to model context primitives for each context feature [Jar10].

[4] `http://www.omg.org/spec/BPMN/2.0/`

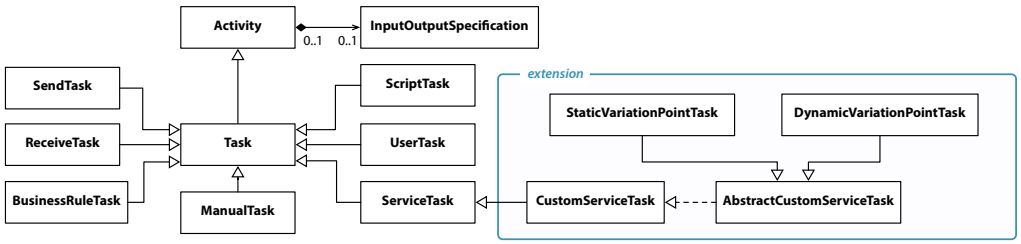


Figure 5.5: Simplified task metamodel of activity extension.

In LateVa, this can be realized by combining context and feature modeling as a unified context variability representation which can be later used for configuring process variants [Mur14a]. We describe variability in a feature tree format (using FMs) where non-context features, context features and constraints are placed. In particular, we make use of Clafer syntax [Bak11] to represent process variability. See [Bak13] for more details on Clafer concrete syntax.

5.3 Design Phase: Application Engineering

At the meta-level, metamodels allow defining either concerns or variations in the operational semantics. In order to realize process variability, in this section we define the application engineering as a systematic procedure consisting of four phases as illustrated in Figure 5.6: (i) *process variability modeling* to create process family implementation artifacts, (ii) *context modeling* to provide a context model that will empower the system to be truly context aware, (iii) *compilation* to obtain platform specific models out of the process variability artifacts, and (iv) *deployment* to store released artifacts onto the Models Repository.

In a nutshell, the procedure follows these steps (see Figure 5.6): first, the developer in charge of modeling, sketches base models, fragments, and variability models. This step is handled together with

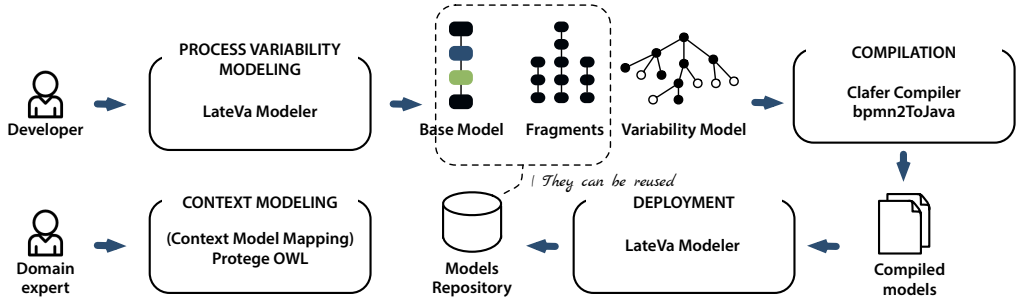


Figure 5.6: Context-aware process variability modeling phases.

the context modeling task, even by the same developer and/or domain expert. Both steps are considered dependent, so that the linkage between a base model, fragments, and a variability model must be valid according to the context description. During compilation phase, we check that the produced artifacts meet all the required syntax rules (syntactical correctness). Finally, deployment consists of pushing process variability artifacts and the corresponding metadata to the Models Repository.

5.3.1 Variability and Context Modeling

The goal of variability and context modeling is to identify process and context variability that may coexist for a process family.

Base Models, Fragments, and Variability Models

At design-time, the LateVa Modeler provides an extended BPMN2 editor to let developers or process designers reuse or define base models and fragments. In a base model, static and dynamic variations can be used in conjunction with standard BPMN2 constructs. Both variation points include a `prefix` attribute for naming, e.g., `S_VP` and `D_VP`; however, the `serviceName` attribute is just provided by

dynamic variation points to indicate specific context data endpoints (e.g., temperature sensor). A fragment may only contain BPMN2 elements. Hence, the granularity of variation points is limited and simplified, inherently incorporated in the base model specification.

The variability associated with variation points and fragments is described within a variability model. LateVa makes use of the Clafer modeling language to describe a FM including: (i) non-context features and features related to process variability constructs (variation point and fragments), (ii) context features that are mapped to both static and dynamic context data key/value pairs, and (iii) cross-tree constraints to represent conditions or restrictions for valid process variant resolution. Features related to variation points and fragments use direct naming compounds (see patterns below). This implies that the same name must appear in base model, variability model or fragment. However, the mapping between context features and context data is defined in the Context Model Mapping.

Pattern for representing static/dynamic variation points

```
{S_,D_} + VP_ + {vpName}
```

Pattern for representing fragments

```
FR_ + {fragmentId} + {parentFeatureName}
```

Context Model and Mapping

LateVa allows two ways of context model representation and mapping: the Context Model Mapping and the ontology+JSON-LD combo.

In the first case, the relationships between context features and context data are described by the Context Model Mapping. This model provides a consistent way of mapping to reveal which context data operate at each context feature in the variability model. As illustrated in Figure 5.7, it defines a *contextVariableName* - a concrete variable name in a domain context model, *featureName* - a context feature or

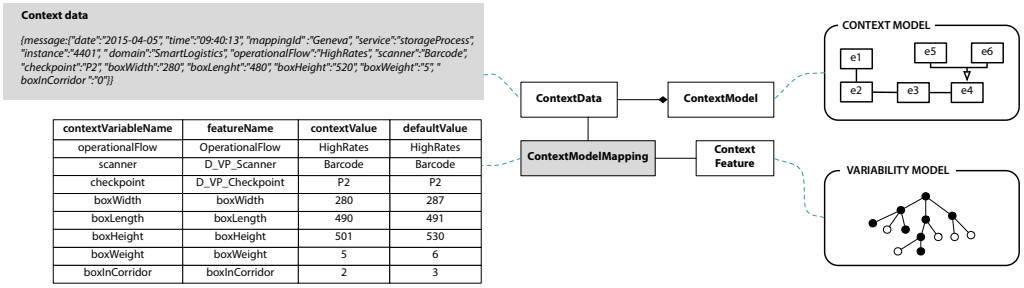


Figure 5.7: Context Model Mapping.

an attribute in a variability model, *contextValue* - the value of a context variable, and *defaultValue* - a valid value which can be assigned for a context variable.

As depicted by Figure 5.7, this mapping model allows to correlate context features with context data; however, this is not a silver bullet as it can be the case where context variables or even context values can be misinterpreted. In other words, different context data may belong to the same context feature partially because context models may differ syntactically and semantically from each other. For instance, while some systems may collect box width in a `boxWidth` variable, `width` can be utilized by others for the same purpose. The terminology might be different and manifold context variable names may belong to the same context feature.

Context Heterogeneity and Ambiguity

Data ambiguity can arise from the same variable names that developers use for different purposes. Such heterogeneity may cause unexpected behavior in context data processing for Context Model Mapping, even allowing identical data values to be sent within different variable names. For instance, it can be difficult to integrate JSON documents from different sources as the data may contain keys that conflict with

other data sources.

A machine doesn't have such an intuitive understanding and sometimes, even for humans, it is difficult to resolve ambiguities in such representations. This problem can be solved by using concise identifiers to denote the different concepts instead of tokens such as "name", etc. Therefore, we need a consistent way of mapping to cope with data ambiguity, incompatibility and inconsistency by defining domain specific syntax and semantics in a non-verbose manner.

As an alternative solution to Context Model Mapping, we proposed another context-FM mapping modeling approach based on Linked Data, i.e., an ontology+JSON-LD combo [Mur14a]. Ontology details domain knowledge and defines the semantics and relationship among context concepts. In contracts, JSON-LD[5] gives a document context, short terminology, and it allows to identify context data to ontology concepts. In [Mur14a], we distinguished between two different ontologies: *data model ontology* and *process context ontology*.

Data model ontology ($Onto_{Data}$) : A context data model describes the data that can be processed and stored including different entity types and their relationships. For instance, each system may have a unique data model (represented by a JSON Schema[6]) and integrated in the data model ontology, where an intentional description of data classes and instances are characterized. Such context information is exploited to correlate the expected contextual data attributes in a variability model to those currently being experienced, as illustrated in Figure 5.8. For example, `RotSpd` data is mapped to ($Onto_{Data}$) data property which also links the `attr1` context feature attribute. Feature attributes are used to map context features, e.g., represented as `integer` values in the diagram above. This way, $Onto_{Data}$ individuals data properties are mapped to context variability model attributes.

[5] `http://json-ld.org`

[6] `http://json-schema.org`

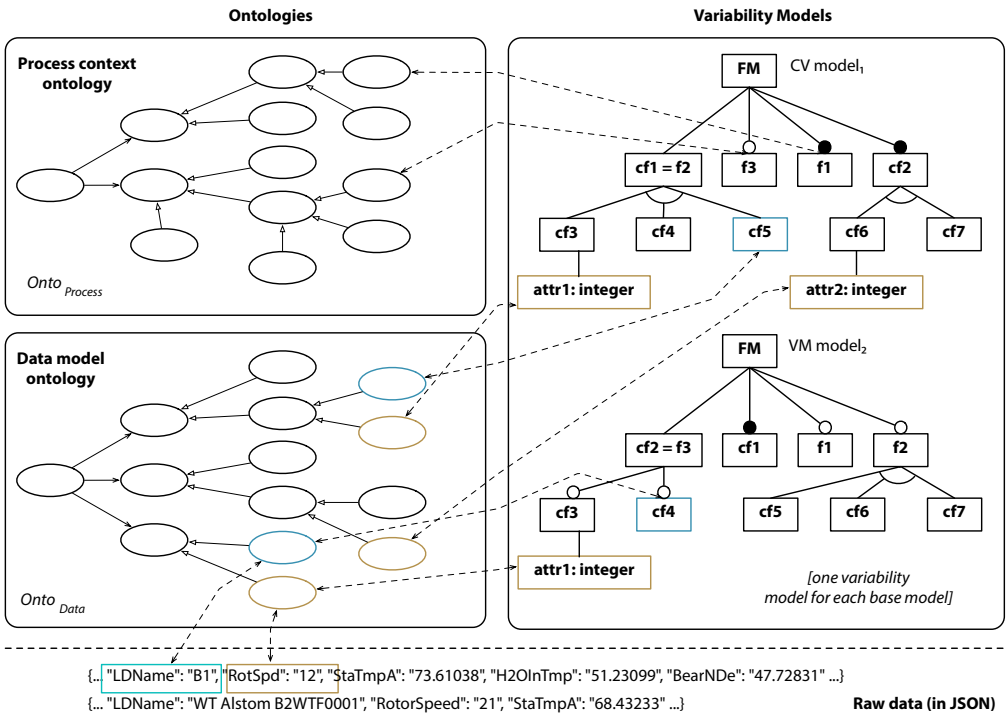


Figure 5.8: Context variability modeling.

On the other hand, $Onto_{Data}$ model is also employed to link piece of raw data coming from the data realm to individuals and data properties of the ontology. Two syntactically context data variable/value pairs (e.g., by two different JSON Schemes) can be semantically equivalent and thus mapped with the same $Onto_{Data}$ concept. For instance, while `RotSpd` can be associated to $CVmodel_2$ and `RotorSpeed` to $CVmodel_1$, both look for the same data property in the $Onto_{Data}$. In a similar vein, considering the `LDName` static context data from the the example above, this could be related to a context feature `B1` which will be activated when the value is met. For that purpose, we adopt the JSON-LD standard which permits to correlate different data variable/value pairs to an equivalent ontology concept, as shown

```
1 {
2   "@context":
3   {
4       "LDName": {
5           "@id": "http://alstom.com/ontoData/windTurbine",
6           "@type": "@id"
7       },
8       "RotSpd": "http://alstom.com/ontoData/wt/rotorSpeed",
9       "StaTmpA": "http://alstom.com/ontoData/wt/generatorTmp",
10      "H2OInTmp": "http://alstom.com/ontoData/wt/waterTmpInlet",
11      "BearNDe": "http://alstom.com/ontoData/wt/nonDriveAndBearingTmp",
12  },
13  "LDName": "B1",
14  "RotSpd": "12",
15  "StaTmpA": "73.61038",
16  "H2OInTmp": "51.23099",
17  "BearNDe": "47.72831"
18  }
```

Figure 5.9: Excerpt of a JSON-LD inline context data.

in Figure 5.9. In that excerpt, context data (e.g., `RotSpd` in line 14) is mapped to ontology concepts (e.g., `ontoData/wt/rotorSpeed` in line 8) so that we are able to preserve semantic meaning. This allows to disambiguate data variables shared among JSON documents from different contexts (e.g. system A, system B) by mapping them to Internationalized Resource Identifiers (IRIs) via an ontology. Thus, standards-based different machine interpretable raw data from various data realms is mapped to ontology concepts (object and data properties), and adequately linked to corresponding context features.

Process context ontology ($Onto_{Process}$) : In conjunction to context data variability management, two syntactically different features from different context variability models can be semantically analogous and thus mapped with the same $Onto_{Process}$ concept. Our $Onto_{Process}$ model defines two main types of primitive classes which include several individuals and object/data properties as follows:

- `Process` *individuals and properties*: We define `BaseModel` and `Fragment` as subclasses of the `Process` class. Individuals belonging to those subclasses are connected by a data property *hasServiceName* to represent service request and also include *hasVariabilityModel* property to `VariabilityModel` individuals. Furthermore, they are linked by data property *hasProcessKey* to deployed base models/fragments in the repository.

- `VariabilityModel` *individuals and properties:* `VariabilityModel` class individuals exemplify different configuration options for each `BaseModel`. They contain a *hasFileName* property to point a particular FM and also a *hasType* for representing the type of FM.

5.3.2 Model Compilation and Deployment

Starting from base models, fragments and variability models, the LateVa Modeler derives a compilation of all these models before deployment. In this step, we make use of the Activiti Engine to translate *.bpmn2 process artifacts into Java objects of `BpmnModel` type. Here, we also validate base models and fragments and check for well-formed structures according to the validator module. On the other hand, the Clafer Compiler checks that all features and constraints within a variability model are well-defined, e.g., checks unique names.

As features related to variation points and fragments use direct naming compounds (i.e., should have the same name) for corresponding base model IDs, the compiler does not check if the inserted base model and fragment references already exist in the Models Repository. Still, such functionality is supported by the LateVa Engine which directly retrieves mapping names and treats exceptions in case of mismatch. After compilation, the developer deploys the produced models to the Models Repository.

5.4 Concluding Remarks

So far, we have presented the domain engineering and application engineering phases that cover the design phase of workflow-based DSPLs. Following the above principles, LateVa solves several issues raised at the beginning of the chapter:

1. **Ensure clean separation of concerns**: LateVa aims for separation of concerns by using BVR as a baseline. Although this approach could limit the visibility of existing variants, it allows users to focus on domain concepts rather than on process elements to efficiently manage large sets of process variants. Moreover, this way it is possible to manage process variability independently reducing its impact over process variants' commonalities, as well as the complexity of managing large sets of process variants.

2. **Promote reuse**: fragments are considered first-class entities in LateVa. They can allow for reuse of process logic among different process variants. At the same time base models can also be reused in the description of a process family. Thus, developers can associate as many variability models as needed with a unique base model, even while aligning fragments to variation points.

3. **Mapping between features, context and process variability artifacts**: context-aware process configuration requires context awareness. In LateVa, this is achieved by aligning FMs and context models through a Context Model Mapping, which gives the correlation between all context features in the FM and context key/value pairs occurred in the context.

4. **Provide context variability support**: since the Context Model Mapping is not preserving data ambiguity and heterogeneity, LateVa may adopt ontologies and JSON-LD for context mapping. The former allows an intentional description of the classes

and entities, while the latter permits to correlate different data key/value pairs to an equivalent ontology concept. This allows to disambiguate data variables shared among JSON documents from different contexts by mapping them to IRIs via an ontology.

In this chapter, we have presented an approach for context-aware process variability modeling and illustrated the process variability realization process combining workflow variability and context modeling techniques. Process variability modeling is guided by the BVR principles and realized through three main models such as base models, fragments, and variability models. Context modeling and reasoning is based on Context Model Mapping and ontologies to map context features. These methods allow to identify dependencies and conflicts between features and context data, and takes such correlations as a basis to derive an appropriate process configuration at run time. All in all, our approach relies on a clean separation of concerns provided by the underlying process variability metamodel.

In the next chapter, we discuss the challenges of deferring process configuration to enable context-aware customization of process variants at run time. Particularly, we emphasize how LateVa engine performs process configuration using the same domain engineering assets described in this chapter.

Chapter 6

Run-time Phase: Context-aware Dynamic Configuration and Execution

In this chapter, we focus on dynamic process configuration. In particular, we describe how to perform context-aware configuration of process variants by means of automatically binding fragments to variation points at execution time. The LateVa engine interprets the models introduced in Chapter 5 and uses a solver to find suitable fragments based on current context data. It also provides extension mechanisms to cope with (un)expected situations, either removing/adding new fragments to a variability model, to support the construction of flexible and variant-rich workflow-based systems.

We next show how these have been used to implement more sophisticated run-time execution and binding.

Structure of the chapter. This chapter begins with several challenges (Section 6.1) for enabling context-aware process configuration at run time. We then present the context-aware configuration life-cycle (Section 6.2), and the internal representation of LateVa engine (Section 6.3). Finally, we revisit the challenges for the run-time phase and summarize the chapter (Section 6.4). This chapter presents and expands upon work published in [Mur14b, Mur14e, Mur14a, Mur15b]

6.1 Run-time Phase Challenges

One of major concerns for process variability is the context-aware configuration of the different variants [Rei14]. This means that context information, not users, may directly influence process configuration for every particular situation. For instance, when context information is only available at run time (dynamic context), the specific variant can be determined during execution. To achieve such type of flexibility and take advantage of the assets developed at the design phase, we identify several challenges as follows:

1. **Run-time variability**: to effectively deal with dynamic situations of the process context, the system should be able to dynamically configure variants at run time. On the other hand, if the process context is already known before execution, process variants should be partially configured before instantiation. This allows for a *staged configuration* which considers static and dynamic context to accomplish effective decision making.

2. **Multiple binding times**: a DSPL demands multiple binding capabilities to select available options/services dynamically. As process context could adapt, e.g., from static to dynamic, the system should change accordingly providing multiple binding times, e.g., from startup to run time. That is, a smooth *transition from one binding to another one* must be guaranteed, so the system may change its binding time without affecting already running instances.

3. **Context awareness and autonomic behavior**: in a context-aware process configuration, the system must handle contextual information to select different options depending on the current conditions on-the-fly. To make this happen, context acquisition and processing should be carried out unobtrusively and continually by each dynamic variation point. Hence, once

context information is collected, the systems should perform *context-aware process configuration* in an automated fashion.

4. **Abstraction support**: context-aware process configuration and run-time decisions may rely not only on contextual information but also on other properties, such as quality levels or user preferences, that may affect the selection of suitable alternatives. As many base models, fragments and data endpoints with different *quality* can be involved in the process configuration, the quality of the overall configuration can vary widely.

6.2 Context-aware Configuration Life-cycle

To better understand the dynamic configuration process, we first describe the context-aware configuration life-cycle for a DSPL. As illustrated in Figure 6.1, the life-cycle consists of four steps:

1. **Context acquisition:** when the base model instance reaches a dynamic variation point execution, the system collects context data from the pre-established context service/API.

2. **Context processing:** once context data is retrieved, such context key/value pairs are mapped to those elements in the variability model, i.e., the relationships between context data and variability features are settled, limiting the scope of the solution space.

3. **Run-time resolution:** based on the altered variability model (context data is already parsed), this step concludes if any sound resolution (fragment) exists for a particular variation point.

4. **Binding and execution:** finally, the system assigns a suitable fragment alternative to the dynamic variation point, which is signaled to re-activate execution and launch a fragment instance.

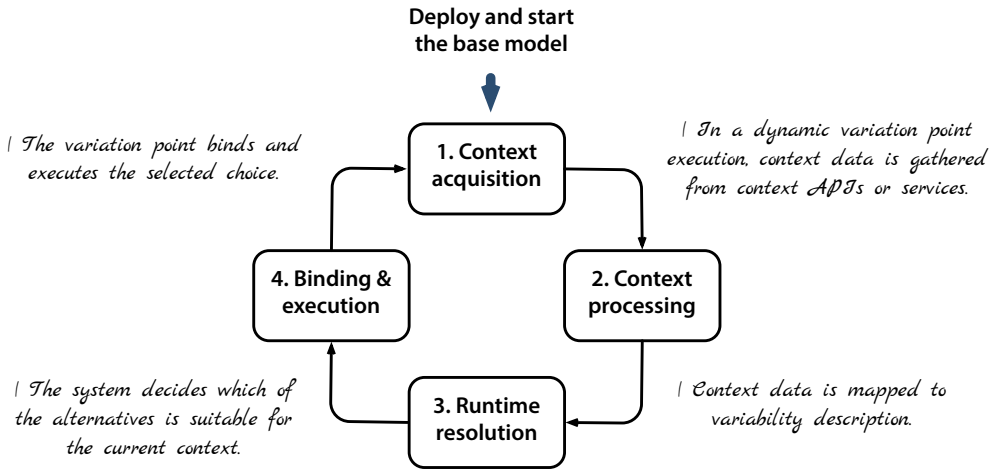


Figure 6.1: Context-aware configuration life-cycle.

We have implemented context-aware process configuration life-cycle in the LateVa engine, which is a prototype framework for context processing and iterative variability resolution.

6.3 Run-time Phase Support

The following section describes in detail how LateVa supports the aforementioned dynamic configuration steps for enabling context awareness, as well as multiple binding times.

6.3.1 Context-aware Dynamic Configuration

As illustrated in Figure 6.2, the LateVa engine starts by receiving a request to trigger a base model execution (Step 1). For each request, the Process Configurator searches for available base models in the Models Repository (RDBMS) that can actually satisfy the user/system requirements (indicated as JSON in Figure 5.7). If more than one base

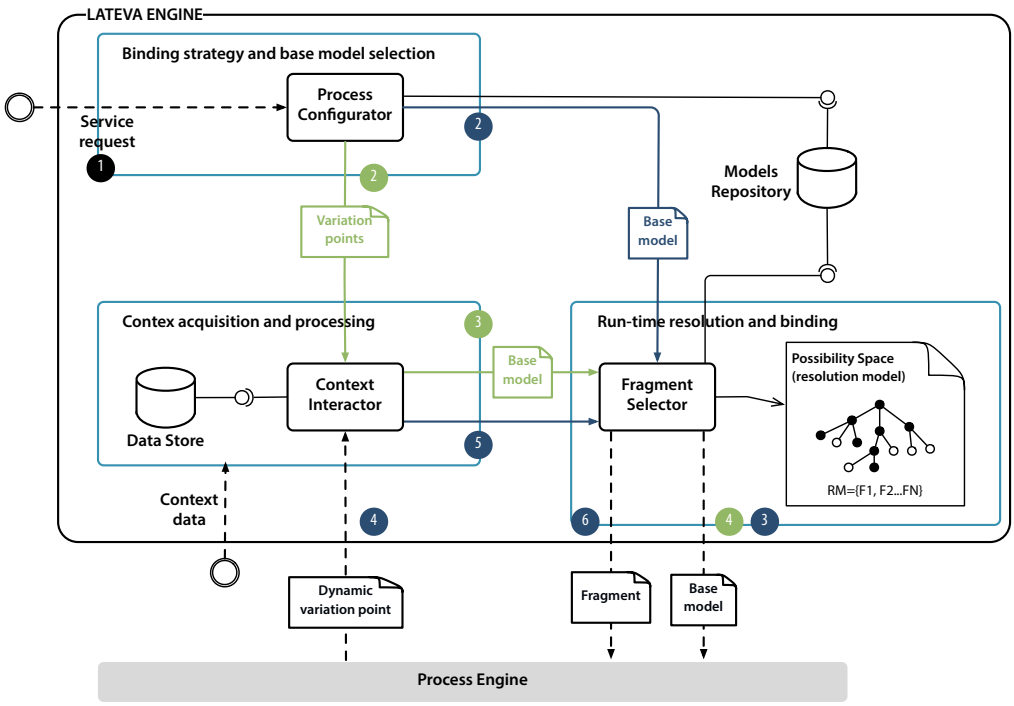


Figure 6.2: Context-aware dynamic configuration.

model exists, the systems gets the first one and locates the attached variability model to bind static context properties.

LateVa enables the transition between multiple binding times. Hence, the user can select a preferred binding time strategy that will change the moment in which variants are bound. Such binding may trigger at different stages of the configuration life-cycle (e.g., configuration-time, deployment-time, startup-time and pure run time/operational mode), selecting suitable alternatives for partially or fully customizing process variants [Cap14a]. LateVa supports multiple binding by allowing only dynamic binding as input, i.e., *startup-time* and *pure run time*. Each of the strategies are marked with one color in Figure 6.2 (green denotes startup-time, and blue denotes pure run time). Both are further described below.

Startup-time Resolution

In the startup binding strategy, the Process Configurator starts by retrieving base model and variability model definitions from the Models Repository. If a base model is found to actually meet the service request (see Step 1 in Figure 6.2), the Context Interactor resolves both static and dynamic variation points (Step 2), and finally the Fragment Selector invokes the base model execution (Steps 3-4), i.e., a base model instance is created in the Models Repository.

It is worth to mention that, even in the startup binding, static variation points do not provide contextual information whereas dynamic variations points involve context interaction. Therefore, for the latter, all context data is stored and parsed before base model instantiation in order to enact context-aware process configuration.

Pure run-time Resolution

In the pure run time binding strategy, the Process Configurator works in a similar fashion than in startup binding, i.e., it fetches a base model and a variability model from the Models Repository (see Step 1 in Figure 6.2) to fulfill the service request. From the selected base model, the Fragment Selector resolves static variation points (Step 2) and runs base model execution (Step 3).

During execution, each base model instance interacts with different services exposed in the context. Such services provide access to existing context data via Context API. Once a base model execution reaches a dynamic variation point (Step 4), the LateVa engine follows the steps depicted in Algorithm 1. For each dynamic variation point, the Context Interactor parses and stores context data to set context features in a variability model (Step 5). This variability model is restored from the Models Repository by the specified base model instance ID. The Fragment Selector uses this altered model as input, and concludes if any sound configuration exists, considering all the solution space, i.e.,

Algorithm 1 Dynamic variation point execution

1: Given: a dynamic variation point y
2: **if** $y_i = fragmentassigned$ **then**
3: Start fragment instance from y_i;
4: **else**
5: Take applicable data endpoint d_s;
6: Invoke d_s to get JSON data j_i and map to the context model;
7: Get one applicable fragment $f_{s,1}$, $f_{s,2}$, ..., f_{s,j_i} for y_i;
8: **end if**
9: **return** fragment f_i for y_i;

considering all the constraints of a variability model. At this point, the Fragment Selector may handle three different *resolution situations*:

- **No solution:** apart from fragment binding, the Fragment Selector must deal with unexpected situations, such as no solution found after model solving. Pro tem, the Fragment Selector rollbacks all context mappings for an unexpected context message, and assists user decision-making.

- **One solution:** to handle this situation, the Fragment Selector assigns fragments for all dynamic variation points from the resulting solution, saves this partial model (resolution model) in the Models Repository, and enacts a base model instance (Step 6). This behavior allows to ensure a proper resolution by considering all the decision space. Finally, when the execution reaches a particular dynamic variation point, fragment execution starts.

- **N-solutions:** when the Clafer solver[1] returns more than one valid solution for the current context, the Fragment Selector may operate by using two different resolution strategies: (a) *get-first*

[1] `https://github.com/gsdlab/chocosolver`

to get the first feasible fragment, [2] and (b) *manual-selection* to enable user-based decision-making. After selecting a fragment for a dynamic variation point by the preferred resolution strategy (Step 6), if just one configuration exists, the remaining dynamic variation points are also solved by assigning a particular fragment.

Following the above strategy, LateVa allows for a staged configuration and execution of context-aware process variability at run time, which can perform fragment binding subsequently based on static and dynamic context.

6.3.2 Quality-of-Result driven Dynamic Configuration

Although workflows have been used to compose and execute a series of computational or data manipulation steps, such in scientific workflows [Alt04, Hau11], a few discussions have been focused on the utilization of run-time mechanisms to configure context-aware process variants based on user-defined quality attributes.

In LateVa, quality is an important concern as many base models, fragments and data endpoints (context services) with different quality accuracy can be involved in the process configuration, so the quality of the overall configuration can vary widely and gets uncertain. For instance, there can be more than one base model that satisfy a given service request or diverse context services with for context data. In this way, LateVa may optimize the process configuration at run time by applying Quality of Result (QoR) metrics, as in [Mur14e].

Regarding abstraction support in process configuration, a requirements driven approach [Lap07] enables the configuration of BPEL processes based on quality constraints. Similarly, a questionnaire-driven approach [LR07] provides a step-wise configuration of reference processes at design-time. However, to

[2]Since Clafer does not support choosing the branding strategy, we cannot tune the solver to control the order of how possible solutions (instances) are generated.

the best of our knowledge, no framework is capable of customizing QoR-driven configuration of process variants run time.

Quality of Result Model (QoRM)

The Quality of Result Model (QoRM) is reflected through a UML diagram, containing intents representing service requests, and constraints QoR representing restrictions, see Figure 6.3 (a).

`QualityOfResult` can determine not only the selection of a particular base model, but also the binding of inherent fragments and context services. A `Constraint` is a key aspect of the QoRM, that represents some condition, restriction or assertion related to the context-aware process variability artifacts. It includes a set of `Conditions` that are atomic formulae or implications. In a simplified version of the 6.3 (a) diagram, three constraints are considered: (i) `Time` for the entire time (`ExecutionTime`) that a fragment or context service takes for execution and its network channel to ping (`ResponseTime`), (ii) `Cost` which represents the cumulative expected cost of performing action, and (iii) `DataQuality` to exhibit the `Availability` and `Accuracy` of provided context services.

In [Mur14e], we defined the domain model using a single ontology (see Figure 6.3 (b)), containing six types of primitive classes which include several individuals and object/data properties as follows: (i) `Intent` individuals with *hasIntentName* data property mapped to the *name* parameter in Figure 6.3, (ii) `Scope` individuals with *hasConfigurationScope* and *hasInteractionScope* data properties, (iii) `Time` , `Cost` and `DataQuality` subclasses of `QualityOfResult` class, (iv) `BaseModel` and `Fragment` as subclasses of `Process` individuals, (v) `DataEndpoint` individuals which contain *hasURI*, *hasServiceName* and *hasDataModel* data properties, and (vi) `ConfigurationModel` individuals with *hasFileName* property to point a particular variability model. Hence, the Fragment Selector is capable of retrieving base models for a given petition and then a

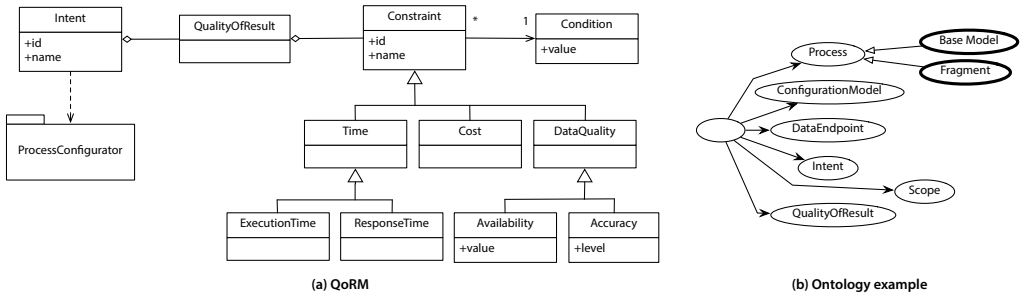


Figure 6.3: (a) A simplified QoR model and (b) sample ontology

base model that meets user-defined QoRM is instantiated.

QoR-driven process configuration (pure run time)

Once a base model execution reaches a dynamic variation point, the process engine follows several steps. If there is no fragment assignment for the current placeholder activity execution, this activity throws an event to select a suitable fragment based on context data. Such selection requires two types of processing. In the first *interaction* task, the event coming from a variation point execution is triggered by the Context Interactor to find a single data endpoint URI that satisfies pre-established QoR constraints (by running a SPARQL query). Data collected from a REST resource (in JSON) is mapped to a data model object (by a *hasDataModel* data property) to automatically perform the base model instance *configuration*. For the latter, the context values gathered from the REST service are mapped to placed attributes in a variability model, in order to get a preferred fragment choice considering pre-established constraints and fragments for each variation point. Once a suitable fragment is resolved using the solver, Fragment Selector signals the particular dynamic variation point execution which executes the preferred fragment.

6.4 Concluding Remarks

After detailing how LateVa supports context-aware dynamic process configuration, let us now revisit the challenges identified at beginning of the chapter and discuss how LateVa faces each of them.

1. **Run-time variability**: LateVa engine enables context-aware dynamic configuration of process variants. This is achieved by resolving dynamic variation points using a pre-established binding strategy. Such resolution involves fragment selection and execution by means of a variability model (expressed in a FM) and Context Model Mapping.

2. **Multiple binding times**: LateVa supports startup and pure run time bindings for enabling workflow-based DSPLs. The former allows to determine static and dynamic variation points before base model instantiation. The latter configures static variation points prior to base model instantiation, and subsequently configures dynamic variation points during execution.

3. **Context awareness and autonomic behavior**: during base model instance execution, dynamic variation points can interact with surrounding context services to collect context data. Such data is parsed to decide which of the available fragments are suitable for the current context. The decision making is realized with automated decision guidance, using Context Model Mapping, a variability model and constraint solving.

4. **Abstraction support**: LateVa has been extended to enable quality-driven selection and configuration. Specifically, our framework enables abstractions to select relevant base models and data endpoints based on user-defined QoR, and provides flexibility in terms of run-time configuration.

In this chapter we have presented the run-time phase for enabling context-aware dynamic configuration of process variants. We have introduced a dynamic configuration life-cycle consisting of four phases such as context acquisition, processing, run-time resolution, and binding and execution. We have also detailed how LateVa supports multiple binding times, including startup and pure run time strategies. To enable dynamic configuration, LateVa makes use of Context Model Mapping and constraint solving to decide which of the available fragment choices fit according to context information. Finally, we have introduced an extension to deal with abstractions in process variability, namely QoR-driven dynamic process configuration.

This chapter concludes the contribution of this dissertation. The next part gives more details on the experimentation and tool support developed for LateVa.

Part III

Validation

"Ideas are easy. Implementation is hard."
-Guy Kawasaki

Chapter 7

Experimental Evaluation

In this chapter, we present the experimentation results using LateVa based on automated warehouse logistics and wind turbine maintenance case studies, which are an extended version of the examples introduced in previous Chapters 5 and 6. We particularly emphasize on the support for run-time variability including the modeling, configuration and enactment phases as well as the different processes that are part of those cases. We also present the tools built around LateVa. This includes a set of modelers, services and APIs for interacting programmatically with the engine. Finally, we discuss the results of the experimentation with a quantitative assessment, as well as the limitations of our framework.

Structure of the chapter. The remainder of this chapter is organized as follows. We first describe the automated warehouse logistics scenario and our experimentation with the case study for storage operational process family (Section 7.1). As a second case study, we outline a wind turbine maintenance process family from a wind turbine manufacturer. Then, we describe the tools developed around the LateVa framework (Section 7.2). Finally, we present a discussion on the advantages and limitations of our approach (Section 7.3) and conclude the chapter (Section 7.4). This chapter presents and expands upon work published in [Mur15a].

7.1 Experimentation

In this section, we intend to extend the LateVa framework to two industrial case studies to provide validation of the proposed approach.

7.1.1 Case Study 1: Automated Warehouse Logistics

In the first case study based on [Mur14b], we assessed the framework's effectiveness through a proof-of-concept storage warehouse operational process family borrowed from our logistics partner ULMA[1]. ULMA provides goods handling systems with a range of solutions for automated warehouses and storage, baggage handling, sorting systems, and picking facilities. It designs and develops ad-hoc integrated logistic services and completes handling installations for each customer, along with training and support.

Warehouse solutions commonly adopt a Warehouse Management System (WMS) to control the movement of materials, stock management and order fulfillment. Along the supply chain, the role of a WMS is to improve the supply chain efficiencies and reduce total running costs of supply chain operations. For that purpose, a WMS controls the receipt, storage and movement of materials to intermediate storage locations or even to third party logistics, and manages the associated transactions, including shipping, receiving, putaway and picking. In order to monitor the progress of goods through stages of the warehouse, a WMS interacts with *automated identification and data capture systems*, such as barcode scanners, sensors, wireless LANs, and Radio-frequency Identification (RFID), as well as *industrial control systems*, such as SCADA and Programmable Logic Controller (PLC) systems.

An *automated warehouse* is a facility where all or some of the tasks related to storing, retrieving and moving inventory are carried out by automated systems. Goods in the warehouse are tagged to allow them

[1] `http://www.ulmahandling.com/en/`

to be located by software systems (e.g., WMS) so that the inventory can be continuously updated as goods move in, out, and around the warehouse. This is achieved by automated identification and data capture systems like RFID. Hence, once data is collected, there can be either a batch synchronization and/or a real-time transmission to a datastore, or even such data can report the status of goods in the warehouse to running operational processes and monitoring tools.

An *operational process* encompasses a group of structured and identifiable activities that contribute to moving a specified and measurable object inside an automated warehouse. As mentioned before, each automated warehouse can be complex and comprises, therefore, different operational processes, such as storage, retrieval and picking. These are further described below:

- **Storage process**: allows goods to be stored based on different location search strategies detailed in a WMS, for example in terms of fixed location, next empty location, and storage unit type.
- **Putaway process**: enables the complete removal of goods from a warehouse location following disparate extractions strategies, for example in terms of FIFO, LIFO, least quantity and expiration date. Extracted goods are then moved to an intermediate area for custom shipping configurations.
- **Picking process**: merges both storage and putaway processes. It consists of collecting products in a specified quantity to satisfy customer orders. After each picking operation, the transport unit (e.g., pallet, cardboard box) with its remaining products is routed back to a disused warehouse location.

Each of the those operational processes may differ from the preceding one, influenced in various ways by warehouse types, storage areas and sensors, but may also expose similarities, which can, to some degree, actually work in similar conditions. Under the current

practice, developers can be tempted to reuse some kind of existing process model parts, but normally operational processes are modeled from scratch for each new warehouse installation. It is therefore clear that such operational processes are likely to be susceptible to errors, for instance in determining the correct activity or control-flow. In a similar vein, designing ad-hoc operational processes may also become a time-consuming and arduous task [Der12, Böh13].

Process reuse can reduce the complexity of managing large collections of similar process variants; however, operational processes should offer greater flexibility so that a WMS can make appropriate decision making in near real-time fashion. For instance, depending on context data coming from presence sensors, the WMS can decide on suitable movements to ensure high-rates on picking operations. This means that we cannot predefine which branches of the control-flow of a running process should be activated due to the unpredictability of upcoming PLC and sensor data. It is therefore crucial to pick up events just-in-time to decide which of the pre-defined flows must be activated. Otherwise, inadequate event processing might have a negative impact on the day to day warehouse operation, often requiring manual intervention (e.g., intrusive process reconfiguration).

Problem: ULMA wants to manage the variability of operational process families (storage, putaway, and picking) considering context data at run time.
Solution: Apply LateVa to model and manage operational process families (storage, putaway, and picking), starting from automated warehouse storage processes. An adequate support in modeling such variant-rich operational processes enhances reuse of variable parts of context-aware operational processes and, by that, may promise an improvement of efficiency during development.

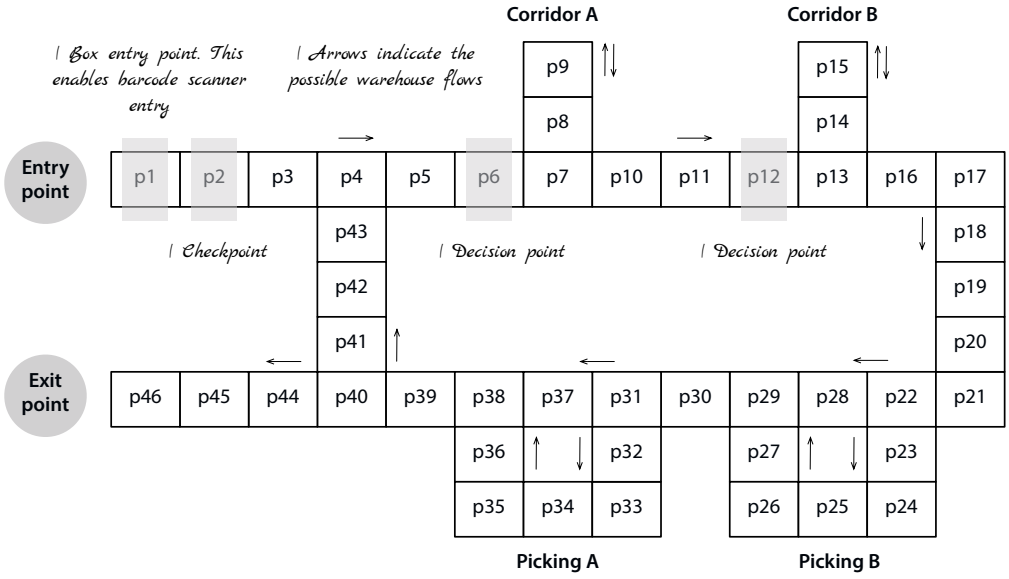


Figure 7.1: Automated warehouse scenario.

Storage Operational Process Family

To provide a first step into this direction, the storage operation process case study analyses the applicability of LateVa as one means to improve the process variability in the automated warehouse logistics domain.

In the proposed case scenario (see Figure 7.1), the warehouse consists of a single material entry point where goods are packed in cardboard boxes, two corridors, 2 picking workstations and a single material exit point. Once the box is registered in the WMS and checked as well-formed box according to pre-established location dimensions, the box is placed in `p1` (each `p` indicates a point in the conveyor system) to initiate the storage workflow. This entrance may take place through complex material entry strategies (RFID, barcode scanner and manual entry) in each WMS (`S_VP_Scanner` in Figure 7.2). As soon as `p1` scanning is completed, the box moves forward to `p2` where a loading gauge checkpoint operates.

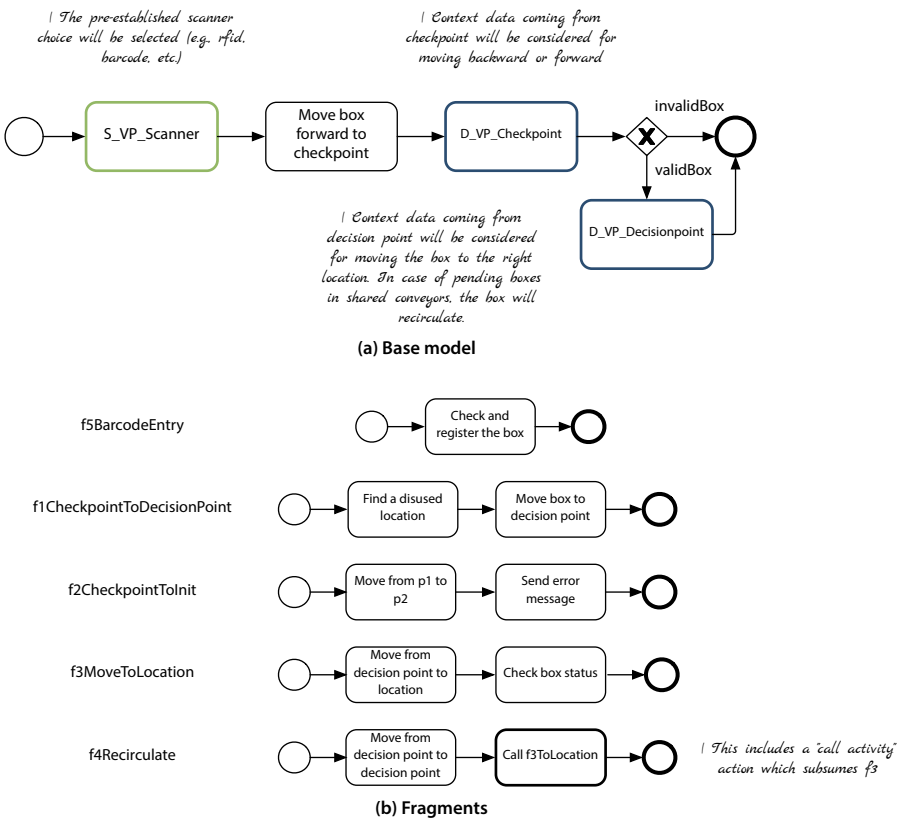


Figure 7.2: (a) Base model and (b) fragments for storage operational process family.

At p2, the framework gathers checkpoint data to decide whether the box can be routed to the physical warehouse location (D_VP_Checkpoint in Figure 7.2). If not, it dispatches a move-backward activity to put the box back at p1. Otherwise, if LateVa concludes that the box is qualified by being located inside the warehouse, the storage process continues its execution up to a decision point where it would be forced to prevent an operation block (D_VP_Decisionpoint in Figure 7.2). In that case, the system will decide to route the box to a fixed location or recirculate it in case of pending boxes in a corridor.

```
AutomatedWarehouse                                  // Global constraints
    xor OperationalFlow                             [Basic  => !FR_f4Recirculate_Decisionpoint]
        HighRates                                   [FR_f1CheckpointToDecisionpoint_P2 => D_VP_Decisionpoint]
        Basic                                       [FR_f2CheckpointToInit_P2 => !D_VP_Decisionpoint]
    xor D_VP_Scanner
        // different box entry alternatives
        RFID
        Barcode
            FR_f5BarcodeEntry_Scanner
        // simple barcode func. Other entry modes are omitted
        Manual
    Layout
        D_VP_Checkpoint
            xor P2
                FR_f1CheckpointToDecisionpoint_P2   // everything ok, move forward
                    boxWidth : integer              // in mm
                        [this < 288]
                    boxLenght : integer             // in mm
                        [this < 492]
                    boxHeight : integer             // in mm
                        [this < 531]
                    boxWeight : integer             // in kg
                        [this < 7]
                FR_f2CheckpointToInit_P2            // in case of any error, move backward
        xor D_VP_Decisionpoint ?
            FR_f4Recirculate_Decisionpoint
                boxInCorridor : integer             // number of in-transit boxes expected by the scheduler
                    [this > 1]
            FR_f3MoveToLocation_Decisionpoint
```

Figure 7.3: Clafer model for storage operational process family.

The storage workflow reflects a typical scenario in which packaged goods are stored into an automated warehouse based on different location search strategies. Figure 7.2 presents the storage process's main activities, containing one static variation point (colored in green) and two variation points (colored in blue). It orchestrates the whole process to track and control all warehouse flows, and enables interaction between physical devices (e.g., conveyor systems, pick to light systems, RFID, and presence sensors, etc.), and warehouse operators (e.g., maintenance manager, workstation agents, and picking operation workers). This allows monitoring of current, operational process and warehouse status, as well as complex event triggering from installed sensors and PLC/SCADA systems.

An automated warehouse is founded on one simple premise: it must guarantee a high-rate operational flow so that storage, retrieval and picking workflows should be aware of existing boxes in shared conveyors. Intending to produce high-rates, presence sensors must

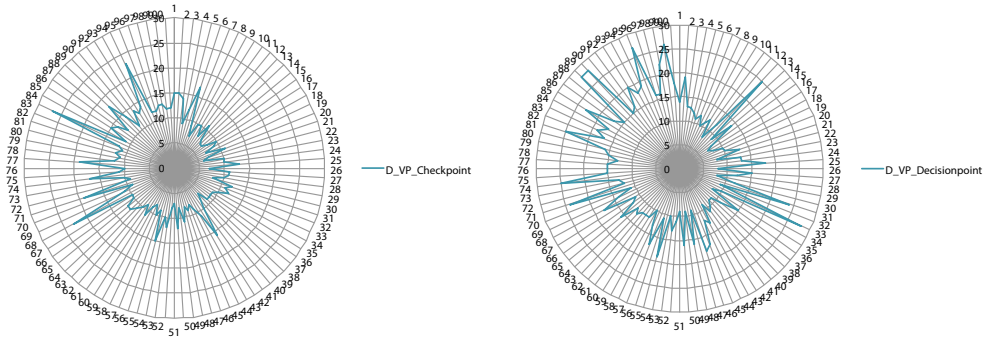
indicate the number of total boxes in such shared conveyor systems in order to ensure that a given threshold is not exceeded. In this light, LateVa constantly communicates with the scheduler, monitors processing movements and coordinates operations. For instance, as illustrated in Figure 7.3, in case of no pending boxes leaving the warehouse locations, the entry box is routed back to a fixed location by executing the right fragment (f3MoveToLocation).

Preliminary Results and Discussion

As depicted by Figures 7.2 and 7.3, we generated 1 base model with 3 variation points (one static and two dynamic), 5 fragments and 1 variability model with 22 features and 8 constraints, covering $3x2x2$ = 12 process variants. These models were used to generate a test case for 100 storage operations. We ran the experiment as a standalone application on a 13inch MacBook Air with 8GB 1600 MHz DDR3 RAM, and Core i7 running @2 GHz.

Figure 7.4 shows the minimum, maximum, and average time taken by each dynamic variation point before fragment execution (pure run time strategy). Consequently, the average Time-to-resolution (TTR) that a base model instance has to wait for using this kind of variation points is 12.83 ms. The delay can be omitted by including events and exclusive gateways in the modeling phase. However, in this case study, we demonstrate the feasibility of applying LateVa to manage dynamic process variability in an automated warehouse logistics domain. This solves several issues raised at the beginning:

- **Reuse**: this is achieved through separation of concerns, i.e., by separating commonality and variability specification of process variability into disjointed models. The key model (variability model) captures variability at the domain level in which variant-rich operational storage processes have been assembled.
- **Context awareness**: the variability model may contain two



min: 6 ms | **max**: 27 ms | **avg**: 11.24 ms | **diff**: 21 ms

min: 7 ms | **max**: 28 ms | **avg**: 14.42 ms | **diff**: 21 ms

Figure 7.4: TTR of dynamic variation points for case scenario 1.

types of domain features: features for domain abstraction and context features related to context data. Dynamic context data coming from PLC/SCADA systems and installed sensors can be controlled by establishing constraints and automatically customizing process variants.

Disclaimer: The proposed synthetic use case has not been tested on a real industrial environment. All context interactions have been simulated using synthetic RESTful services. Context Model Mapping was adopted for context mapping.

7.1.2 Case Study 2: Wind Turbine Maintenance

The second case scenario presented in this section has been prepared at the scope of work with Alstom[2], which means to automatically integrate information from different wind farm data sources into Operation and Maintenance (O&M) processes. Alstom is one of the

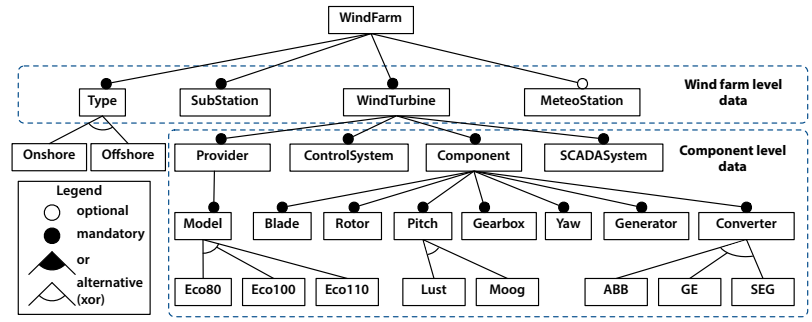
[2] `http://www.alstom.com/`

world's leading energy solutions and rail transport manufacturer, active in the fields of passenger transportation, signaling and locomotives, with products including the AGV, TGV, Eurostar, and Pendolino high-speed trains, in addition to suburban, regional and metro trains, among others. Alstom power activities include the design, manufacturing, and supply of products and systems for the power generation sector and industrial markets. The group covers most of the energy sources, including gas, coal, nuclear, hydro and wind; the latter is our focus here.

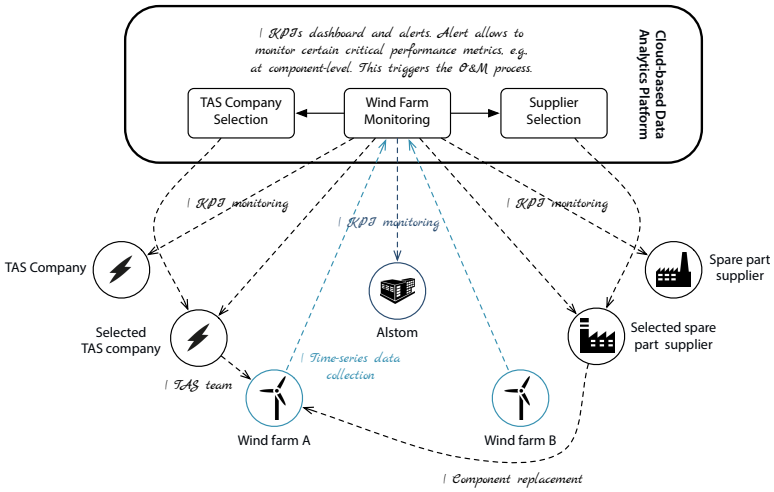
Wind farms regularly incorporate sub-stations to provide a software-defined protection for wind farm grid connection and may also have meteo-stations for weather forecasting. At wind farm level, multiple stakeholders such as Original Equipment Manufacturers (OEMs), business owners, maintenance manager and other third parties may store, process, analyze and access real-time and historical data from different gateways of installed PLC/SCADA systems, as well as sub-stations and meteo-stations. Beyond the particulars of different wind turbine providers, an onshore/offshore wind turbine consists of a number of components sourced from OEMs, which offer hardware that can be assembled to form a convenient power generation machine.

A simplified version of a wind turbine is shown in Figure 7.5 (a), where seven main components are considered: (i) wind turbine `Blades`, (ii) a `Pitch` for controlling blade's speed, (iii) a `Rotor` for integrating blades, (iv) a `Gearbox` increases blades' speed, (v) a `Yaw` for controlling the rotation of the tower, (vi) an electricity `Generator`, and (vii) a volt AC/DC `Converter`. Apart from high-level data provided by wind farms, such components (each having an individual data model) also emit fine-grained component-specific raw data that can be accessed by different stakeholders of wind energy domain to get meaningful insights by structural health monitoring (SHM), condition monitoring predictive failure analysis (PFA), operation status, and real-time monitoring such as power curve monitoring [Sch13].

Data services are globally dispersed under control of Alstom and



(a) A simplified wind farm ecosystem



(b) Wind farm scenario

Figure 7.5: (a) A simplified wind farm ecosystem and (b) the wind farm scenario.

ready to be used whenever needed. Atop of such services, a number of process models may exist which orchestrate domain-specific activities related to aforementioned services and deal with a huge data streams for analytics. When a Key Performance Indicator (KPI) goes into an "alert" state, the Cloud platform triggers a O&M process to check the status of a particular wind farm and proceed with maintenance work.

An *O&M process* includes a set of activities to efficient wind turbine components maintenance. To that aim, it encompasses

interconnected companies along its supply chain (see Figure 7.5 (b)), from onshore/offshore wind farms where wind turbines are installed, to spare parts suppliers and certified Technical Assistance Service (TAS) companies which perform different types of interventions such as periodic maintenances or breakdown repairs.

Wind turbine components incorporate high levels of quality and reliability due to the harsh conditions they are subjected to, thus requiring high availability of manufacturing assets in order to reduce cost of production downtimes and the potential impact on delivery times. To tackle this issue, Alstom offers its customers a technical maintenance and repair service by means of a O&M process to minimize the number of unscheduled downtimes and reduce wind turbine breakdowns. However, considering the different types of wind farms, components, maintenance contracts and stakeholders, it turns out that designing ad-hoc O&M processes for a particular scenario is not efficient; therefore, variability should be treated and managed. This would lead to a reduction in O&M process modeling time, as well as an increase of efficiency and performance while in operation.

Problem: Alstom wants to manage the variability of its O&M processes considering context data coming from wind farms.
Solution: Apply LateVa to model and manage O&M process families, considering two of the components such as rotor blades and yaw system. This enhances reuse of existing O&M processes, as well as an intelligent decision making on suitable spare parts suppliers based on context data. Hence, Alstom expect to obtain a reduction of unscheduled machine downtimes on customer's plants and improvements of managing the interventions by managing dynamic variability of its own O&M processes.

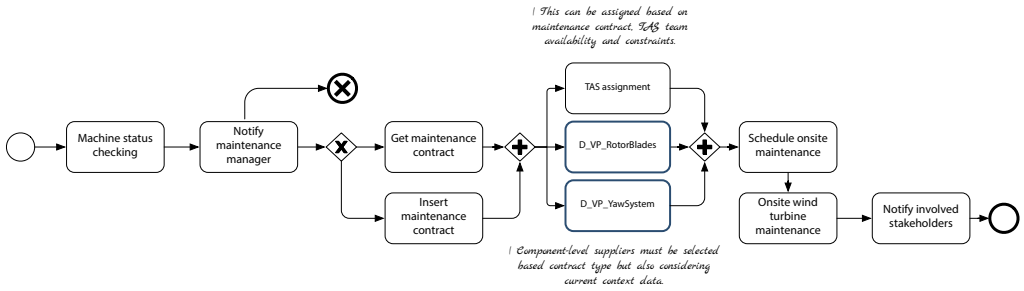


Figure 7.6: Base model for wind farm O&M process process family.

Wind Turbine Maintenance Process Family

As illustrated by Figure 7.6, in the event that a component wear or failure in a wind turbine is detected, the systems triggers the O&M base model to determine the actual health status of wind farm components and evaluate alternatives in case of error.

In case of a real wind turbine error, the O&M base model checks the maintenance agreement and the workflow continues by collecting component-level data to determine the degree of severity of the error and select a suitable spare part supplier, as well as a convenient TAS team. In Figure 7.6 two dynamic variation points (`D_VP_RotorBlades` and `D_VP_YawSystem` in Figure 7.6) have been placed in order to consider those components that are potentially vulnerable to errors [Tch14]. However, this base model can be expanded to consider all the components shown in Figure 7.5 (a).

The variability model (see Figure 7.7) shows selectable fragment alternatives, context features and corresponding constraints. In this case, we add fake context variables to yaw and rotor blades features.

Apart from component-level context values, aspects such as the location of the referred wind farm, spare parts suppliers and TAS companies, availability of spare parts or capacity in suppliers and skilled staff, intervention costs and response times can also be taken

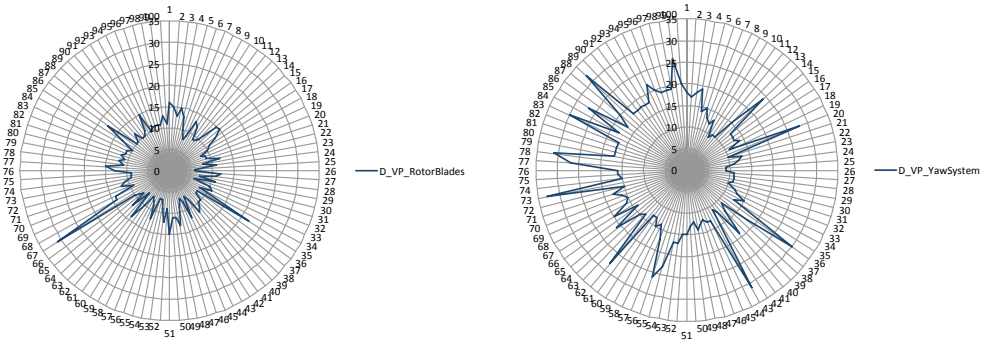
```
WindTurbine
        xor Type
                Offshore
                Onshore
        xor MaintenanceContract
                NonStop
                Basic
        Component
                D_VP_YawSystem
                        xor Y01 ?
                                FR_f1_Y01
                                        yw200 : integer                         // yw200 < 90
                                                [this < 90]
                                FR_f3_Y01
                                        ytd5 : integer                          // ytd5 > 6
                                                [this > 6]
                                        ytd7 : integer                          // ytd7 > 8
                                                [this > 8]
                                        ytd9 : integer                          // ytd9 < 34
                                                [this < 34]
                                        ytd11 : integer                         // ytd11 < 22
                                                [this < 22]
                                FR_f4_Y01
                        D_VP_RotorBlades
                                xor RB01 ?
                                FR_f2_RB01
                                FR_f5_RB01
                                        fr1010 : integer                        // fr1010 in {300..400}
                                                [this > 300 && this < 400]
                                        fr1011 : integer
                                                [this > 250]                    // fr1011 > 250
                                        fr1012 : integer
                                                [this > 180]                    // fr1012 > 180

// Global constraints
[FR_f4_Y01 => FR_f2_RB01]  // F4 implies F2
[Basic => !FR_f3_Y01]      // Basic excludes F3
```

Figure 7.7: Clafer model for O&M wind farm process family.

into account in run-time decision-making.

After assigning the spare parts supplier for each component and the TAS company commissioned to perform the work, LateVa starts the fragment execution.[3] This will facilitate the coordination of the process reporting events such as the spare part shipping to the customer and its arrival at the corresponding wind farm. In this way, the selected TAS company will properly manage their staff to perform the work at the right time. Upon completion of the intervention, it will be checked that the new component correctly emits time series to the Alstom Cloud platform.

[3]In this example we just created synthetic fragments due to the lack of real component-level maintenance information.



min: 6 ms | **max**: 31 ms | **avg**: 10.95 ms | **diff**: 25 ms

min: 9 ms | **max**: 33 ms | **avg**: 16.67 ms | **diff**: 24 ms

Figure 7.8: TTR of dynamic variation points for case scenario 2.

Preliminary Results and Discussion

In this case study we have employed 2 dynamic variation points, 5 synthetic fragments, and a variability containing 25 features and 10 constraints. As in the previous case scenario, we generated a test case for 100 base model executions and ran the experiments as a standalone application on a 13inch MacBook Air with 8GB 1600 MHz DDR3 RAM, and Core i7 running @2 GHz.

Figure 7.8 shows the minimum, maximum, and average time taken by each dynamic variation point before fragment execution (pure run time strategy). The average TTR that a base model instance has to wait for using dynamic variation points in a parallel gateway is 13.81 ms. Even having such a reasonable TTR, we believe that this can be improved by using asynchronous activity execution.

In this case study, given the magnitude of the problem, the adoption of LateVa would have a major impact on the involved stakeholders, wind farm customers, Alstom and spare part suppliers, e.g., as is intended in the CREMA project[4]. The presented use case might be

[4] `http://www.crema-project.eu`

extended to other critical wind turbine components, increasing the current service level and maintenance monitoring. The four issues raised in this work are resolved as follows:

- **Reuse**: component-level fragments can be reused along different wind farms. This way O&M processes can share pre-established process structures, avoiding unnecessary ad-hoc modeling.
- **Context awareness**: data from PLC/SCADA systems can be gathered in a real-time fashion in order to consider which of the available options is the most suitable.

Disclaimer: The proposed synthetic use case has not been tested on a real industrial environment. We used synthetic RESTful context services and fragments to test the base model execution, as well as for performance measures. Context Model Mapping was adopted for context mapping.

7.2 Tool Support

Automation within DSPLs aims at reducing the load of work on developers by identifying repetitive tasks that do not need human intervention and can be automatized. Usually, in workflow-based systems these tools are in charge of reusing exiting parts that are common and can be factorized from different workflow scenarios. Such tools allows developers to focus on domain requirements and thus to spend less time solving common problems, such as in defining and managing context-aware and variant-rich warehouse operational processes or wind farm O&M processes. LateVa intents to provide a set of tools that can be used at different stages of the process life-cycle [Wes12], from the modeling and configuration, and going all the way down to the execution and binding of processes at run time.

```
1  public interface LatevaEngineAPI {
2
3     BaseModel deployBaseModel(String baseModelName);
4
5     List<BaseModel> listBaseModels();
6
7     FragmentModel deployFragment(String fragmentName);
8
9     List<FragmentModel> listFragments();
10
11    VariabilityModel deployVariabilityModel(String varModelName, long basemodelId);
12
13    List<VariabilityModel> listVariabilityModels();
14
15    void submit(String jsonRequest, Properties props);
16 }
```

Figure 7.9: LateVa API interfaces.

In order to have a better understanding of the tools, in this section, we briefly describe the set of tools implemented around LateVa.

7.2.1 LateVa Modeler

The LateVa modeler is an extension of the Activiti Designer[5] which includes variability-specific constructs such as static and dynamic variation points inside "LateVa Extension". These tasks implement their own behavior, as described in process variability metamodel (see Figure 5.2) and placed in `lateva-modeler-ui` module.

7.2.2 LateVa Engine

The LateVa engine includes the three services mentioned in Chapter 6 such as Process Configurator, Context Interactor, and Fragment Selector. These were implemented in the `lateva-engine` module.

7.2.3 LateVa API

Java API

LateVa API provides programmatic access to deploy and read base models, fragments and variability models among others. Figure 7.9 summaries the set of interfaces to represents functions in LateVa's API.

More examples and LateVa source code can be found on Github: `https://github.com/amurguzur/lateva`

7.3 Discussion

In this section we present a quantitative evaluation of the LateVa framework in terms of computation time and scalability, as well as enumerate the limitations and weaknesses of our toolkit.

7.3.1 Quantitative Analysis

This section presents a quantitative evaluation of the LateVa framework to understand how the run-time variability may impact the standard process instance execution. Through an empirical study and comparable evaluation metrics, we evaluate the scalability and computation time (performance) of the framework using a synthetic scenario. We do not consider QoR during configuration and we employ Context Model Mapping mechanism; however QoR-driven configuration using ontologies was evaluated in [Mur14e].

Experimental Setup

For the evaluation, we created a synthetic base model as a baseline (`C1` in Table 7.1) which consists of 4 dynamic variation points (`D_VP_act1`

[5] We followed the steps described in `http://www.activiti.org/userguide`

and D_VP_act2 sequentially, and D_VP_act3 and D_VP_act4 in a parallel gateway) and the corresponding control flow. Each variation point comprises 4 fragment choices. Variability is described in a feature model with 21 features, 8 attributes (2 for each variation point, producing 8 context features) and 8 constraints. Hence, in C1 we obtain in total $4^4 = 256$ possible variants. Additionally, we created another eleven cases and grouped all them into three main sets:

1. *C1-C4 increasing number of fragments (Set1)*: represents progressively larger number of fragments for each variation point, from 4 (C1) to 32 fragments (C4).

2. *C5-C8 increasing number of data mappings (Set2)*: represents progressively larger number of context features, from 16 (C5) to 128 (C8), all having Integer values.

3. *C9-C12 increasing number of variation points (Set3)*: represents progressively larger number of variation points in the base model, from 5 (C9) to 8 (C12).

Factors

For Sets (1-3), we distinguished three different configuration factors that may impact scalability and performance metrics when dealing with large sets of process variants:

1. *Number of fragments (Fra)*: measures how the increasing number of fragments per variation point may influence system performance.

2. *Number of context features (Con)*: shows the overall number of attributes that correlate context variable/value pairs in the variability model.

Table 7.1: Aggregated evaluation results (TTR for Min, Max and Avg in ms)

Set	**Case**	**Factor**			**Metric (TTR of 500 runs)**			
		Fra	Con	Var	Min	Max	Avg	*# of variants*
Set1	*C1*	4	8	4	9	20	10.977	256
	C2	8	8	4	9	65	12.079	4,096
	C3	16	8	4	10	53	12.595	65,536
	C4	32	8	4	12	61	15.118	1,048,576
Set2	*C5*	4	16	4	9	55	11.790	256
	C6	4	32	4	10	54	13.483	256
	C7	4	64	4	13	68	15.923	256
	C8	4	128	4	17	72	21.730	256
Set3	*C9*	4	10	5	9	71	12.457	1,024
	C10	4	12	6	9	69	13.307	4,096
	C11	4	14	7	10	39	14.116	16,384
	C12	4	16	8	10	70	15.312	65,536
							14.074	

3. *Number of variation points (Var)*: supports generalization by checking how the framework operates with an increasing number of variation points.

Metrics

In order to get reliable numbers, the base model was processed 500 times for each evaluation case (provided as a supplement file). Performance metrics were measured by TTR, which indicates the time needed by the engine to reactively configure a singular dynamic variation point.[6] We evaluated the results against the following three quantitative metrics:

1. *Min. TTR (Min)*: minimum TTR for each evaluation case considering all variation point resolutions.

[6]All datasets and the supplement spreadsheet file of the evaluation are available at: `http://git.io/vvDV4`

Table 7.2: Evaluation results (Avg TTR for each variation point in ms)

Set	Case	Metric (Avg TTR)							
		D_VP_act1	D_VP_act2	D_VP_act3	D_VP_act4	D_VP_act5	D_VP_act6	D_VP_act7	D_VP_act8
Set1	*C1*	10.344	10.356	12.224	10.984	-	-	-	-
	C2	11.312	11.438	13.258	12.310	-	-	-	-
	C3	11.824	11.880	13.962	12.716	-	-	-	-
	C4	14.320	14.420	16.512	15.222	-	-	-	-
Set2	*C5*	10.956	11.156	13.150	11.900	-	-	-	-
	C6	12.848	13.018	14.766	13.300	-	-	-	-
	C7	15.014	15.428	17.342	15.910	-	-	-	-
	C8	20.047	21.010	23.332	22.502	-	-	-	-
Set3	*C9*	11.012	11.094	14.732	13.408	12.040	-	-	-
	C10	11.154	11.096	16.398	14.952	13.788	12.454	-	-
	C11	11.244	11.126	17.672	16.570	15.280	14.140	12.780	-
	C12	11.438	11.420	19.734	18.386	17.134	15.962	14.908	13.516

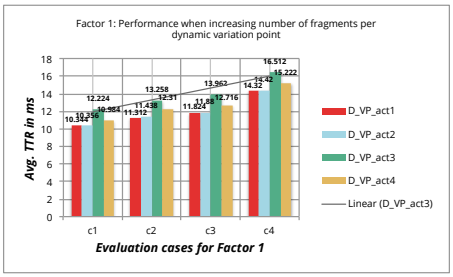
2. *Max. TTR (Max)*: maximum TTR for each evaluation case considering all variation point resolutions.

3. *Avg. TTR (Avg)*: average TTR for each evaluation case considering all variation point resolutions.

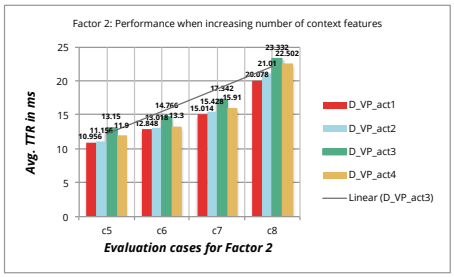
Results

The results of the study in terms of the average number of scenario runs can be found as numbers in Table 7.1 and Table 7.2, and as graphics in Figure 7.10. All the results are given in milliseconds (ms) and numbers, where appropriate. Considering all 500 evaluation runs, Table 7.2 provides the average TTR for each variation point in each `Case (1-12)`.

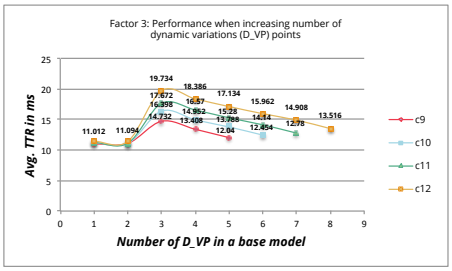
As an aggregated snapshot, Table 7.1 presents the observed metrics (Min, Max and Avg TTR) for all evaluation cases and related factors (Fra, Con and Var). Overall, we may state that the complexity of our framework ***scales linearly with respect to the number of fragments (Fra), context features mapped to context data (Con) and variation points (Var) with little impact on performance***. We argue that this can help to manage large sets of variants (e.g., 1,048,576 in `C4` with 8



(a) Factor 1: Increasing fragments (Fra)



(b) Factor 2: Increasing context features (Con)



(c) Factor 3: Increasing dynamic variation points (Var)

Figure 7.10: Quantitative evaluation results.

context features) that involve a wide variety on context data types (e.g., 128 context features are considered in C8 for 256 variants). Additional evaluation details are given below.

As depicted by Figure 7.10 (a), the increasing number of fragments per variation point slightly affects the average performance for resolving a variation point. For instance, as in Table 7.1, the difference between the average TTR for C1 with 4 fragments and C4, where 32 fragments are placed for each variation point, is about **4.141 ms**. Similarly (see Table 7.2), we can figure out how the resolution of variation points in a parallel gateway (e.g., 16.512 ms for D_VP_act3 and 15.222 ms for D_VP_act4 in C4) is more TTR consuming in comparison to placeholder activities that are coordinated sequentially (e.g., 14.320 ms for D_VP_act1 and 14.42 ms for D_VP_act2 in C4).

As can be extracted from Table 7.1, a minimal performance penalty is obtained when substantially adding more context features to the variability model. The average TTR for processing and configuring

dynamic variation points based on 128 context features is 21.730 ms. If we compare this with C5 (see Figure 7.10 (b)), the average execution time increases to **9.94 ms**.

Regarding TTR when the increasing number of variation points in a base model (see Figure 7.10 (c)), it can be deduced that sequentially modeled placeholder activities do not affect overall processing time (around 11.012 ms and 11.094 ms for C9 in Table 7.2) , whilst additional variation points in a parallel gateway employ more TTR (e.g., 14.732 ms for D_VP_act3 in C9, 16.398 ms in C10, 17.672 ms in inC11, 19.734 ms in C12). Furthermore, due to the current implementation, LateVa needs more time to resolve the first variation point than similar activities in a parallel gateway. For instance, while D_VP_act3 in C9 requires 14.732 ms, it goes down to 12.04 ms for D_VP_act5. In terms of scalability, the difference between C9 and C12 avg. TTR is still minimal (**2.855 ms**).

Using the empirical data we can deduce that even though Set3 metrics are competitive and may constitute an improvement to existing process reconfiguration alternatives (around 300 ms for process model and instance migration in [Bar12] and for model operations in [Alf14]), even for the more complex cases C11 and C12 in this evaluation example, due to the fact that LateVa is built to support multiple binding times such as startup-time and pure run time, this causes its performance to be lower than that of Set1 and Set2.

Threats to Validity

By deferring process variability to run time, separating process variability modeling into disjointed models (i.e., positive variability) and considering context data, the flexibility of provided processes can be improved without the need for human intervention. In LateVa, the time to automatically customize a single variation point is reasonable ($\simeq$ 14 ms) even with a large number of process variants, so that the total time to complete a process instance execution is slightly increased. Therefore, we argue that our framework is efficient when dealing with

large set of process variants (scalability) in context-aware systems, under controlled circumstances (Sets 1-3).

Fragments and variability models can be updated when necessary proving the availability of the system being modeled. The time consumption of the adaptation approach scales linearly, even in case of constraint solving. However, the limited amount of experimental data may influence this result, so it would be interesting to learn more about the usefulness of the framework in a real-life scenario.

7.3.2 Limitations

LateVa is not a panacea. We have developed a prototype framework to demonstrate the feasibility of the proposed solution. In the following lines, we discuss the limitations of the approach both from the point of the implementation and from the applicability of LateVa.

1. **Modeling effort**: the base models, fragments, variability models and context mappings used in the empirical study were just examples. We do not have enough evidence about the effort required during the modeling. Nevertheless, we argue that LateVa would not exponentially increase human effort during the modeling phase since this phase is fully tool-supported. We expect to quantify this effort as well as validate the metrics of the empirical study in real world scenarios.

2. **Activiti and Clafer dependent**: although LateVa implements the process variability metamodel presented in Figure 5.2, the framework is tight to Activiti and Clafer. For BPMN processes, we extended the Activiti framework with special constructs that could be reused in other BPMN2 compliant framework such as Camunda[7].

[7] `https://camunda.com/`

3. **Limited context data types**: in the experimental evaluation we employed Integer data types as context data. This was mainly because Clafer only supported such data type (now supports strings). We should also consider more data types (strings, reals, etc.) as context features in the variability model specification.

4. **Limited context interaction**: LateVa only allows RESTful services to be consumed on context interaction. However, this could be extended to support other protocols such as SOAP services, TCP, MQTT or more.

5. **Lack of real-world evaluation**: we provided a quantitative evaluation, but we did not have any real-world analysis. The lack of real-world evaluation is a common problem in the field of process variability [Ayo15], partially because the lack of accessibility to real-world processes and datasets.

7.4 Summary

In this chapter we presented the experimentation of the LateVa framework. The purpose of both scenarios, i.e., automated warehouse logistics and wind turbines, was to determine the potential applicability of LateVa in industrial workflow-based systems. We also presented in detail the tools built around LateVa, including the LateVa modeler, LateVa engine, and an API. The last part of the chapter focused on a quantitative evaluation and a justification for the choices made in the framework. For the evaluation we have discussed how our approach stands regarding performance and scalability using a synthetic scenario.

This chapter concludes the third part of this dissertation dedicated to the validation of the LateVa framework. In the next chapter we summarize the main contributions, draw conclusions, and define a set of perspectives for future work.

Part IV

Conclusion

"Your time is limited, so don't waste it living someone else's life"
-Steve Jobs

Chapter 8

Conclusion

How should we design and execute variant-rich workflows for the new era of data-intensive systems? This dissertation shows that the answer can be quite simple: a context-aware dynamic variability modeling and execution approach, based on FMs and constraint solving, can achieve a reasonable computation time in a wide range process families, offering an efficient and automated resolution (process configuration) mechanism at run time.

While process (variability) modeling and workflow execution engines will undoubtedly evolve, we hope that the LateVa framework proposed here can, at very least, offer a useful reference point. The LateVa framework is currently only 3600 lines of code, and models built on it are anecdotal examples rather than real-world workflows. In an area where application requirements and context changes are fast evolving, we believe that run-time variability management, over the difficult part of the large variability that some particular systems may exhibit, can enable greater flexibility in process configuration.

Even if LateVa has not been tested in real-world industrial settings, we believe that its main contribution is to enable context-aware process configuration of process variants at run time. As workflow-based systems grow in complexity, they will need to manage process variability in order to facilitate reuse and to raise the level of abstraction for future workflows and frequently used control-flow. The use of

process variability in today's dynamic environments is considerably more difficult than the widely studied static problem, partially due to changing context data. In LateVa, dynamic variability of such workflows solves this problem by enabling process configuration at run time. At the modeling phase, separation of concerns allows base models, fragments, and variability models to efficiently coexist, improving developer's modeling effort ("design by reuse"). At run time, the engine can resolve variability in an automated fashion with a minor performance penalty.

Structure of the chapter. In the rest of this chapter, we summarize a few of the lessons that influenced this work (Section 8.1). Finally, we sketch areas for short-term/long-term future work (Section 8.2).

8.1 Lessons Learned

The importance of separation of concerns. One interesting lesson is how to model process variability, i.e., decide between negative and positive modeling. In many cases, negative has been used to encompass all process variants in a single configurable process model which actually provides major visibility of all possible alternatives compared to positive variability. However, as it was pointed out in the comparison in Chapter 3, positive modeling is taking hype over the last years. Why is that? This is partially because it allows to model commonality and variability is separated models and thus is "scale free".

Essentially, this is an interesting way to approach large-scale process families because it means that variants can be created on-demand without affecting ongoing ones. For example, LateVa gives developers the ability to create base models and fragments within the domain engineering phase to reuse existing models in the application engineering phase. During execution, new fragments can also be created to replace existing ones, without requiring users to choose

manually which alternative is the right for current context.

Flexibility over complexity. While it can be sufficient to configure process variants at design-time (using either configuration or deployment binding), another lesson from our work is that real-world workflow-based systems are often more complex, and it is flexibility in these complex systems that matters. In particular:

- Most process variants will *combine context data* in data-intensive environments, e.g., parsing context data from underlying sensors, services or APIs, and them using this data to decide on the appropriate flow-routing.
- Most process variants will *claim flexibility* in context-aware systems, requiring run-time and automated execution models that behave well under dynamic conditions.

For example, suppose that a process variant created at design-time, e.g., using a configurable process model, was updated during execution. The specialized system might interrupt the process execution at the end of each activity that involves intrusive monitoring capabilities to detect potential changes, and thus might incur performance penalties.

We believe that due to the unpredictable nature of dynamic environments, context data will continue to be important, requiring software systems to defer decisions to run time, as in DSPLs. In these types of environments, we believe that run-time variability engines will have to be optimized for an efficient execution and provide flexibility benefits.

Applicability and performance. One interesting lesson is how to design an engine that does not look bottleneck. In many cases, deferring decisions to run time might limit the performance of the

workflow-based system, so most of the efforts have been put on design-time variability. For example, Provop [Hal10], Template & Rules [Kum12] and vBPMN [Döh13] enable process configuration at design-time. Why is that? These approaches assume that in a controlled environment produced process variants will rarely change over time.

This type of variability works best in static environments, but not in dynamic settings where context changes and user requirements may constantly evolve. We believe that LateVa could be applicable in latter conditions while performance will not be unnecessarily compromised.

8.2 Perspectives

Even if the results obtained from this research can be considered consequent, there are still several areas that can be further explored.

8.2.1 Scope and Short-term Perspectives

Several areas are still open for improvement. Here below we present some of the points that would improve LateVa in the short term.

Correctness checking. In LateVa, we have analyzed our framework for correctness support including syntactical correctness but not behavioral correctness. The former guarantees the correct structure of the customized models, e.g., avoiding disconnected models. The latter guarantees the correct behavior of the customized models to ensure process instance completion, e.g., by avoiding deadlocks and livelocks. Since our framework configures process variants at run time, at least the behavioral correctness of each individual customization a posteriori can be difficult, as it was reported in [LR13]. However, behavioral correctness support would provide a more exceptional solution dealing with data-flow correctness and providing a guidance

to avoid livelocks during execution. This would be a good point for the near future.

Support for more data types and sophisticated models. As was briefly discussed in Chapter 7, the evaluation only considered Integer data types as context values. This was because Clafer solver only allowed for Integer attributes. It would be desirable to move on to test more data types and validate evaluation results with a more extensive collection of data types, as well as more sophisticated models (base models, fragments, and variability models).

Monitoring UI. Although "Process Explorer" was initially planned in [Mur13b], LateVa does not support this feature yet. Such monitoring capabilities would provide access to the LateVa engine and its API for process administration, including: (i) variability management to manage and change pre-defined variability models, (ii) instance inspection to inspect and filter details of running process instances, and (iii) statistical historic data analysis to supervise historical context data. The latter might also encompass data ingestion and processing.

REST API. The REST API will decouple monitoring UI and third-party application integration. Therefore, clients of the REST API will use the supplied variability aliases to talk about process variability, and will be unaware of the connection and location details. The same interfaces and resources from Java API can be used, so the implementation of REST API is fairly straightforward.

Framework agnostic. Although the mentioned API interfaces are framework agnostic, the various implementations of these interfaces are framework specific (Activiti and Clafer). This could be expanded to consider other frameworks and/or languages. It would mean developers with framework-agnostic LateVa version that would be

able to quickly get up and running process variants using the desired platform such as Activiti, Camunda, Apache ODE, Clafer, CVL, etc.

Experimental but real-life evaluation. The presented evaluation was based on synthetic models. It would be beneficial to evaluate the applicability of the proposed framework in real-life situations. This would require to access and analyze existing workflows and inherent context data, to successfully model process and context variability in more elaborated models.

8.2.2 Long-term Perspectives

In order to further improve LateVa, we also sought additional perspectives in the long-term.

Support multi-perspective configuration. Although we moved in such direction [Mur13c, Mur14c], we consider that it would be interesting to analyze how multi-perspectives process configuration can be achieved while preserving separation of concerns and run-time support. This means that not only functional perspective will influence configuration, but also other process perspectives (e.g., organizational or informational) described in Chapter 2.

Recommendations-driven configuration. Current Fragment Selector only supports two different resolution strategies such as get-first and manual selection. As context data acquisition is very important when designing context aware systems, it can get the same importance while in process configuration. In that particular case, the system could adopt process mining techniques [Aal10] to recommend possible applicable fragments on the basis of actual context data and historical fragment/context choices.

Formal aspects for approach reuse. Several parts of the proposed approach could be enriched and improved by applying well founded theoretical studies. For instance, we believe that it could be interesting to further formalize the mapping between features and context data, as well as the metamodel showed in Chapter 5. This would allow us to express our approach in a formal way and reuse and/or adapt in different application contexts.

Bibliography

[Aal04a] Wil MP van der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management. In *Lectures on concurrency and Petri nets*, pp. 1–65. Springer, 2004.

[Aal04b] Wil MP van der Aalst and Kees Max Van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004.

[Aal10] Wil MP van der Aalst, Maja Pesic, and Minseok Song. Beyond process mining: From the past to present and future. In *CAiSE*, pp. 38–52. Springer, 2010.

[Aal13] Wil MP van der Aalst. Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013.

[Abo99] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pp. 304–307. Springer, 1999.

[Ach12] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B France. Separation of concerns in feature modeling: support and applications. In *AOSD*, pp. 1–12. ACM, 2012.

[Ada06] Michael Adams, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *CoopIS*, pp. 291–308. 2006.

[Ada07] Michael Adams, Arthur HM Ter Hofstede, Wil MP van der Aalst, and David Edmond. Dynamic, extensible and context-aware exception handling for workflows. In *CoopIS*, pp. 95–112. 2007.

[Aie10] Marco Aiello, Pavel Bulanov, and Heerko Groefsema. Requirements and tools for variability management. In *COMPSAC Workshops*, pp. 245–250. IEEE, 2010.

[Alf14] Germán H Alférez, Vicente Pelechano, Raúl Mazo, Camille Salinesi, and Daniel Diaz. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*, vol. 91:pp. 24–47, 2014.

[Alt04] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM*, pp. 423–424. IEEE, 2004.

[Ard11] Danilo Ardagna, Luciano Baresi, Sara Comai, Marco Comuzzi, and Barbara Pernici. A service-based framework for flexible business processes. *IEEE software*, vol. 28(2):pp. 61–67, 2011.

[Asa14] Mohsen Asadi. *Developing and Validating Customizable Process Models*. Ph.D. thesis, Communication, Art & Technology: School of Interactive Arts and Technology, 2014.

[Ayo12a] Clara Ayora, Victoria Torres, Vicente Pelechano, and Germán H Alférez. Applying cvl to business process

variability management. In *MODELS Workshops*, pp. 26–31. ACM, 2012.

[Ayo12b] Clara Ayora, Victoria Torres, Manfred Reichert, Barbara Weber, and Vicente Pelechano. Towards run-time flexibility for process families: Open issues and research challenges. In *BPM Workshops*, pp. 477–488. Springer, 2012.

[Ayo15] Clara Ayora, Victoria Torres, Barbara Weber, Manfred Reichert, and Vicente Pelechano. Vivace: A framework for the systematic evaluation of variability support in process-aware information systems. *Information and Software Technology*, vol. 57:pp. 248–276, 2015.

[Bak11] Kacper Bak, Kacperk, Krzysztof Czarnecki, and Andrzej Wasowski. Feature and meta-models in clafer: Mixed, specialized, and coupled. In *Software Language Engineering*, pp. 102–122. Springer, 2011.

[Bak13] Kacper Bak. *Modeling and analysis of software product line variability in Clafer*. Ph.D. thesis, University of Waterloo, 2013.

[Bal10] Soren Balko, Arthur HM Ter Hofstede, Alistair P Barros, Marcello La Rosa, and Michael J Adams. Business process extensibility. *EMISA*, vol. 5(3):pp. 4–23, 2010.

[Bar12] Luciano Baresi, Sam Guinea, and Liliana Pasquale. Service-oriented dynamic software product lines. *Computer*, vol. 45(10):pp. 42–48, 2012.

[Bar14] Luciano Baresi, Sam Guinea, and Valerio Panzica La Manna. Consistent runtime evolution of service-based business processes. In *WICSA*, pp. 77–86. IEEE, 2014.

[Bar15] Luciano Baresi and Clément Quinton. Dynamically evolving the structural variability of dynamic software product lines. In *SEAMS*, p. 7. 2015.

[Bay06] Joachim Bayer, Sebastien Gerard, Øystein Haugen, Jason Mansell, Birger Møller-Pedersen, Jon Oldevik, Patrick Tessier, Jean-Philippe Thibault, and Tanya Widen. Consolidated product line variability modeling. In *SPLC*, pp. 195–241. Springer, 2006.

[Ben10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, vol. 35(6):pp. 615–636, 2010.

[Ber12] Thorsten Berger. Variability modeling in the wild. In *SPLC*, pp. 233–241. ACM, 2012.

[Ber13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. A survey of variability modeling in industrial practice. In *VaMoS*, p. 7. ACM, 2013.

[Bet10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, vol. 6(2):pp. 161–180, 2010.

[Böh13] Martin Böhmer, Damian Daniluk, Michael Schmidt, and Heiko Gsell. Business object model for realization of individual business processes in the logistics domain. In *Efficiency and Logistics*, pp. 237–244. Springer, 2013.

[Bos12] Jan Bosch and Rafael Capilla. Dynamic variability in software-intensive embedded system families. *Computer*, (10):pp. 28–35, 2012.

[Bur10] Thomas Burkhart and Peter Loos. Flexible business processes-evaluation of current approaches. In *Multikonferenz Wirtschaftsinformatik*, p. 243. 2010.

[Can08] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. A framework for qos-aware binding and re-binding of composite web services. *Journal of Systems and Software*, vol. 81(10):pp. 1754–1769, 2008.

[Cap11] Rafael Capilla and Jan Bosch. The promise and challenge of runtime variability. *Computer*, vol. 44(12):pp. 93–95, 2011.

[Cap13] Rafael Capilla, Jan Bosch, and Kyo-Chul Kang. *Systems and Software Variability Management*. Springer, 2013.

[Cap14a] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Cortés, and Mike Hinchey. An overview of dynamic software product line architectures and techniques: Observations from research and industry. *Journal of Systems and Software*, vol. 91:pp. 3–23, 2014.

[Cap14b] Rafael Capilla, Oscar Ortiz, and Mike Hinchey. Context variability for context-aware systems. *Computer*, vol. 47(2):pp. 85–87, 2014.

[Cap15] Rafael Capilla, Mike Hinchey, and Francisco J Díaz. Collaborative context features for critical systems. In *VaMoS*, p. 43. ACM, 2015.

[Cet08] Carlos Cetina, Vicente Pelechano, Pablo Trinidad, and Antonio Ruiz Cortés. An architectural discussion on dspl. In *SPLC*, pp. 59–68. 2008.

[Cet09a] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer*, vol. 42(10):pp. 37–43, 2009.

[Cet09b] Carlos Cetina, Øystein Haugen, Xiaorui Zhang, Franck Fleurey, and Vicente Pelechano. Strategies for variability transformation at run-time. In *SPLC*, pp. 61–70. Carnegie Mellon University, 2009.

[Cet10] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Designing and prototyping dynamic software product lines: Techniques and guidelines. In *SPLC*, pp. 331–345. Springer, 2010.

[Cha07a] Soo Ho Chang and Soo Dong Kim. A variability modeling method for adaptable services in service-oriented computing. In *SPLC*, pp. 261–268. IEEE, 2007.

[Cha07b] Anis Charfi and Mira Mezini. Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, vol. 10(3):pp. 309–344, 2007.

[Cha09] Anis Charfi, Tom Dinkelaker, and Mira Mezini. A plug-in architecture for self-adaptive web service compositions. In *ICWS*, pp. 35–42. IEEE, 2009.

[Cle02] Paul Clements and Linda Northrop. *Software product lines: Practices and patterns*. Addison-Wesley, 2002.

[Cub11] Javier Cubo and Ernesto Pimentel. Damasco: A framework for the automatic composition of component-based and service-oriented architectures. In *ECSA*, pp. 388–404. Springer, 2011.

[Cub13] Javier Cubo, Nadia Gamez, Lidia Fuentes, and Ernesto Pimentel. Composition and self-adaptation of service-based systems with feature models. In *ICSR*, pp. 326–342. Springer, 2013.

[Cza04] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In *SPLC*, pp. 266–283. Springer, 2004.

[Cza12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wasowski. Cool features and tough decisions: A comparison of variability modeling approaches. In *VaMoS*, pp. 173–182. ACM, 2012.

[Der12] Wassim Derguech, Feng Gao, and Sami Bhiri. Configurable process models for logistics case study for customs clearance processes. In *BPM Workshops*, pp. 119–130. Springer, 2012.

[Dij12] Remco M Dijkman, Marcello La Rosa, and Hajo A Reijers. Managing large collections of business process models-current techniques and challenges. *Computers in Industry*, vol. 63(2):pp. 91–97, 2012.

[Döh11] Markus Döhring and Birgit Zimmermann. vbpmn: Event-aware workflow variants by weaving bpmn2 and business rules. In *Enterprise, Business-Process and Information Systems Modeling*, pp. 332–341. Springer, 2011.

[Döh13] Markus Döhring. *Handling Variants and Adaptation along the Life Cycle of Event-Aware Workflows*. Ph.D. thesis, University of Jena, 2013.

[Döh14] Markus Döhring, Hajo A Reijers, and Sergey Smirnov. Configuration vs. adaptation for business process variant maintenance: An empirical study. *Information Systems*, vol. 39:pp. 108–133, 2014.

[Dum13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. *Fundamentals of business process management*. Springer, 2013.

[Eka11] Chathura C Ekanayake, Marcello La Rosa, Arthur HM Ter Hofstede, and Marie-Christine Fauvet. Fragment-based version management for repositories of business process models. In *OTM*, pp. 20–37. Springer, 2011.

[Fan12] Marcelo Fantinato, Maria Beatriz Felgar de Toledo, Lucinéia Heloisa Thom, Itana Maria de Souza Gimenes, Roberto dos Santos Rocha, and Diego Zuquim Guimarães Garcia. A survey on reuse in the business process management domain. *International Journal of Business Process Integration and Management*, vol. 6(1):pp. 52–76, 2012.

[Gam15] Nadia Gamez, Joyce El Haddad, and Lidia Fuentes. Managing the variability in the transactional services selection. In *VaMoS*, p. 88. ACM, 2015.

[Gin10] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Developing mobile workflow support in the internet of things. *IEEE Pervasive Computing*, vol. 9(2):pp. 18–26, 2010.

[Got08] Florian Gottschalk, Wil MP van der Aalst, Monique H Jansen-Vullers, and Marcello La Rosa. Configurable workflow models. *International Journal of Cooperative Information Systems*, vol. 17(02):pp. 177–221, 2008.

[Gra13] Gregor Grambow, Nicolas Mundbrod, Vivian Steller, and Manfred Reichert. Challenges of applying adaptive processes to enable variability in sustainability data collection. In *SIMPDA*. CEUR-WS. org, 2013.

[Gro14] Katarina Grolinger, Miriam AM Capretz, Americo Cunha, and Said Tazi. Integration of business process modeling and web services: A survey. In *SOCA*, vol. 8, pp. 105–128. Springer, 2014.

[Gun08] Christian W Gunther, Stefanie Rinderle-Ma, Manfred Reichert, and Wil MP van der Aalst. Using process mining to learn from process changes in evolutionary systems. *International Journal of Business Process Integration and Management*, vol. 3(1):pp. 61–78, 2008.

[Hal08a] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Context-based configuration of process variants. In *TCoB*. 2008.

[Hal08b] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, vol. 41(4):pp. 93–95, 2008.

[Hal10] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing variability in business process models: The provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22(6-7):pp. 519–546, 2010.

[Har08] Herman Hartmann and Tim Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *SPLC*, pp. 12–21. IEEE, 2008.

[Hau08] Øystein Haugen, Birger Moller-Pedersen, Jon Oldevik, Gøran K Olsen, and Andreas Svendsen. Adding standardized variability to domain specific languages. In *SPLC*, pp. 139–148. IEEE, 2008.

[Hau11] Matheus Hauder, Yolanda Gil, and Yan Liu. A framework for efficient data analytics through automatic configuration

and customization of scientific workflows. In *e-Science*, pp. 379–386. IEEE, 2011.

[Hau12] Øystein Haugen, Andrzej Wasowski, and Krzysztof Czarnecki. Cvl: Common variability language. In *SPLC*, pp. 266–267. ACM, 2012.

[Hau14] Øystein Haugen and Ommund Øgård. Bvr-better variability results. In *System Analysis and Modeling: Models and Reusability*, pp. 1–15. Springer, 2014.

[Hei99] Petra Heinl, Stefan Horn, Stefan Jablonski, Jens Neeb, Katrin Stein, and Michael Teschke. A comprehensive approach to flexibility in workflow management systems. In *ACM SIGSOFT Software Engineering Notes*, vol. 24, pp. 79–88. ACM, 1999.

[Her10] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. Using complex event processing for dynamic business process adaptation. In *SCC*, pp. 466–473. IEEE, 2010.

[Jar10] Zakwan Jaroucheh, Xiaodong Liu, and Sally Smith. Mapping features to context information: Supporting context variability for context-aware pervasive applications. In *WI-IAT*, vol. 1, pp. 611–614. IEEE, 2010.

[Kan90] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Tech. rep., DTIC Document, 1990.

[Kap10] Malinda Kapuruge, Jun Han, and Alan Colman. Support for business process flexibility in service compositions: An evaluative survey. In *ASWEC*, pp. 97–106. IEEE, 2010.

[Kar09] Dimka Karastoyanova and Frank Leymann. Bpel'n'aspects: Adapting service orchestration logic. In *ICWS*, pp. 222–229. IEEE, 2009.

[Kar14] Ahmet Serkan Karataş and Halit Oğuztüzün. Attribute-based variability in feature models. *Requirements Engineering*, pp. 1–24, 2014.

[Kem11] Sandy Kemsley. The changing nature of work: From structured to unstructured, from controlled to social. In *BPM*, pp. 2–2. Springer, 2011.

[Kon09] Michiel Koning, Chang-ai Sun, Marco Sinnema, and Paris Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Information and Software Technology*, vol. 51(2):pp. 258–269, 2009.

[Kop08] Oliver Kopp, Daniel Martin, Daniel Wutke, and Frank Leymann. On the choice between graph-based and block-structured business process modeling languages. In *MobIS 2008*. Citeseer, 2008.

[Kum12] Akhil Kumar and Wen Yao. Design and management of flexible process variants using templates and rules. *Computers in Industry*, vol. 63(2):pp. 112–130, 2012.

[Lap07] Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-driven design and configuration management of business processes. In *BPM*, pp. 246–261. Springer, 2007.

[Lee12] Jaejoon Lee, Gerald Kotonya, and Daniel Robinson. Engineering service-based dynamic software product lines. *Computer*, (10):pp. 49–55, 2012.

[LR07] Marcello La Rosa, Johannes Lux, Stefan Seidel, Marlon Dumas, and Arthur HM Ter Hofstede. Questionnaire-driven configuration of reference process models. In *CAiSE*, pp. 424–438. Springer, 2007.

[LR11] Marcello La Rosa, Marlon Dumas, Arthur HM Ter Hofstede, and Jan Mendling. Configurable multi-perspective business process models. *Information Systems*, vol. 36(2):pp. 313–340, 2011.

[LR13] Marcello La Rosa, Wil MP van der Aalst, Marlon Dumas, and Fredrik P Milani. Business process variability modeling: A survey. *ACM Computing Surveys*, 2013.

[Lu07] Ruopeng Lu and Shazia Sadiq. A survey of comparative business process modeling approaches. In *BIS*, pp. 82–94. Springer, 2007.

[Mec14] Inbal Mechrez and Iris Reinhartz-Berger. Modeling design-time variability in business processes: Existing support and deficiencies. In *BPMDS*. Springer, 2014.

[Mee11] Stephanie Meerkamm. Configuration of multi-perspectives variants. In *BPM Workshops*, pp. 277–288. Springer, 2011.

[Mee12] Stephanie Meerkamm. Staged configuration of multi-perspectives variants based on a generic data model. In *BPM Workshops*, pp. 326–337. Springer, 2012.

[Met14] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: Achievements and challenges. In *ICSE*, pp. 70–84. ACM, 2014.

[Mil10] Hafedh Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi.

Business process modeling languages: Sorting through the alphabet soup. *ACM Computing Surveys*, vol. 43(1):p. 4, 2010.

[Mon08] Ildefonso Montero, Joaquin Pena, and Antonio Ruiz-Cortés. From feature models to business processes. In *SCC*, vol. 2, pp. 605–608. IEEE, 2008.

[Mor08] Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jézéquel, Arnor Solberg, Vegard Dehlen, and Gordon Blair. An aspect-oriented and model-driven approach for managing dynamic variability. In *MoDELS*, pp. 782–796. Springer, 2008.

[Mor09] Brice Morin, Olivier Barais, J Jezequel, Franck Fleurey, and Arnor Solberg. Models@ run. time to support dynamic adaptation. *Computer*, vol. 42(10):pp. 44–51, 2009.

[Mur12a] Aitor Murguzur. Model-driven and planning for service composition in dynamic heterogeneous environments. In *SummerSOC*. 2012.

[Mur12b] Aitor Murguzur and Goiuria Sagardui. Towards a model-based hybrid service composition for dynamic environments. In *ECSOC*. 2012.

[Mur13a] Aitor Murguzur, Rahim Makhani, Binjiang Tao, and Davide Zambon. Flexible processes and process mining: A brief survey. Tech. rep., IBM, 2013.

[Mur13b] Aitor Murguzur, Goiuria Sagardui, Karmele Intxausti, and Salvador Trujillo. Process variability through automated late selection of fragments. In *CAiSE Workshops*, pp. 371–385. 2013.

[Mur13c] Aitor Murguzur, Hong-Linh Truong, and Schahram Dustdar. Multi-perspective process variability: A case for smart green buildings. In *SOCA*, pp. 25–29. 2013.

[Mur14a] Aitor Murguzur, Rafael Capilla, Salvador Trujillo, Oscar Ortiz, and Roberto Lopez-Herrejon. Context variability modeling for runtime configuration of service-based dynamic software product lines. In *SPLC Workshops*, pp. 2–9. ACM, 2014.

[Mur14b] Aitor Murguzur, Xabier De Carlos, Salvador Trujillo, and Goiuria Sagardui. Context-aware staged configuration of process variants@runtime. In *CAiSE*, pp. 241–255. 2014.

[Mur14c] Aitor Murguzur, Xabier De Carlos, Salvador Trujillo, and Goiuria Sagardui. On the support of multi-perspective process models variability for smart environments. In *MODELSWARD*, pp. 549–554. 2014.

[Mur14d] Aitor Murguzur, Karmele Intxausti, Aitor Urbieta, Salvador Trujillo, and Goiuria Sagardui. Process flexibility in service orchestration: A systematic literature review. *International Journal of Cooperative Information Systems*, vol. 23(03), 2014.

[Mur14e] Aitor Murguzur, Johannes M. Schleicher, Hong-Linh Truong, Salvador Trujillo, and Schahram Dustdar. Drain: An engine for quality-of-result driven process-based data analytics. In *BPM*, pp. 349–356. Springer, 2014.

[Mur15a] Aitor Murguzur, Salvador Trujillo, and Goiuria Sagardui. Dynamic variability support in workflow-based systems: An evaluation of the lateva framework. In *SAC*. 2015.

[Mur15b] Aitor Murguzur, Salvador Trujillo, Hong-Linh Truong, Schahram Dustdar, Oscar Órtiz, and Goiuria Sagardui.

Runtime variability for context-aware smart workflows. *IEEE Software*, 2015.

[Nur08] Selmin Nurcan. A survey on the flexibility requirements related to business processes and modeling artifacts. In *HICSS*, pp. 378–378. IEEE, 2008.

[Par09] Carlos Parra, Xavier Blanc, and Laurence Duchien. Context awareness for dynamic service-oriented product lines. In *SPLC*, pp. 131–140. Carnegie Mellon University, 2009.

[Pes08] Maja Pesic. *Constraint-based workflow management systems: shifting control to users*. Ph.D. thesis, Technische Universiteit Eindhoven, 2008.

[Pic12] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *BPM Workshops*, pp. 383–394. Springer, 2012.

[Puh05] Frank Puhlmann, Arnd Schnieders, Jens Weiland, and Mathias Weske. Variability mechanisms for process models. *PESOA-Report TR*, vol. 17:pp. 10–61, 2005.

[Rec05] Jan Recker, Michael Rosemann, and Wil MP van der Aalst. On the user perception of configurable reference process models-initial insights. *ACIS*, p. 66, 2005.

[Reg05] Gil Regev and Alain Wegmann. A regulation-based view on business process and supporting system flexibility. In *CAiSE*, vol. 5, pp. 91–98. 2005.

[Reg06a] Gil Regev, Pnina Soffer, and Rainer Schmidt. Taxonomy of flexibility in business processes. In *BPMDS*. 2006.

[Reg06b] Gil Regev and Alain Wegmann. Business process flexibility: Weick's organizational theory to the rescue. In *BPMDS*. 2006.

[Reg07] Gil Regev, Ilia Bider, and Alain Wegmann. Defining business process flexibility with the help of invariants. *Software Process: Improvement and Practice*, vol. 12(1):pp. 65–79, 2007.

[Rei09] Manfred Reichert and Peter Dadam. Enabling adaptive process-aware information systems with adept2. *Handbook of Research on Business Process Modeling*, pp. 173–203, 2009.

[Rei12] Manfred Reichert and Barbara Weber. *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media, 2012.

[Rei14] Manfred Reichert, Alena Hallerbach, and Thomas Bauer. Lifecycle management for business process variants. *Handbook on Business Process Management*, 2014.

[Ros07] Michael Rosemann and Wil MP van der Aalst. A configurable reference modelling language. *Information Systems*, vol. 32(1):pp. 1–23, 2007.

[Rus06] Nick Russell, Wil MP van der Aalst, and Arthur HM Ter Hofstede. Exception handling patterns in process-aware information systems. Tech. rep., 2006.

[Sad01] Shazia Sadiq, Wasim Sadiq, and Maria Orlowska. Pockets of flexibility in workflow specification. In *ER*, pp. 513–526. Springer, 2001.

[Sai14] Oumaima Saidani and Selmin Nurcan. Business process modeling: A multi-perspective approach integrating variability. In *EMMSAD*, pp. 169–183. Springer, 2014.

[Sba14] Hanae Sbai, Mounia Fredj, and Laila Kjiri. A pattern based methodology for evolution management in business process reuse. *International Journal of Computer Scienc*, vol. 11(1), 2014.

[Sch08] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil MP van der Aalst. Process flexibility: A survey of contemporary approaches. In *Advances in Enterprise Engineering I*, pp. 16–30. Springer, 2008.

[Sch11] David Schumm, Dimka Karastoyanova, Frank Leymann, and Steve Strauch. Fragmento: Advanced process fragment library. In *Information Systems Development*, pp. 659–670. Springer, 2011.

[Sch13] Meik Schlechtingen, Ilmar F. Santos, and Sofiane Achiche. Using data-mining approaches for wind turbine power curve monitoring: A comparative study. *Sustainable Energy, IEEE Transactions on*, vol. 4(3):pp. 671–679, July 2013.

[Sin07] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, vol. 49(7):pp. 717–739, 2007.

[SR13] Roberto dos Santos Rocha and Marcelo Fantinato. The use of software product lines for business process management: A systematic literature review. *Information and Software Technology*, vol. 55(8):pp. 1355–1373, 2013.

[Str04] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *UbiComp Workshops*. 2004.

[Tch14] Pierre Tchakoua, René Wamkeue, Mohand Ouhrouche, Fouad Slaoui-Hasnaoui, Tommy Andy Tameghe, and Gabriel Ekemb. Wind turbine condition monitoring:

State-of-the-art review, new trends, and future challenges. *Energies*, vol. 7(4):pp. 2595–2630, 2014.

[Tor12] Victoria Torres, Stefan Zugal, Barbara Weber, Manfred Reichert, Clara Ayora, and Vicente Pelechano. A qualitative comparison of approaches supporting business process variability. In *BPM Workshops*, pp. 560–572. Springer, 2012.

[Val13a] George Valença, Carina Alves, and Vander Alves. Analysing variability management in bpm and spl: A knowledge mapping. In *SBSI*. 2013.

[Val13b] George Valença, Carina Alves, Vander Alves, and Nan Niu. A systematic mapping study on business process variability. *International Journal of Computer Science & Information Technology*, vol. 5(1), 2013.

[Vil14] Karina Villela, Adeline Silva, Tassio Vale, and Eduardo Santana de Almeida. A survey on software variability management approaches. In *SPLC*, pp. 147–156. ACM, 2014.

[Web08] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features–enhancing flexibility in process-aware information systems. *Data & knowledge engineering*, vol. 66(3):pp. 438–466, 2008.

[Web09] Barbara Weber, Shazia Sadiq, and Manfred Reichert. Beyond rigidity–dynamic process lifecycle support. *Computer Science-Research and Development*, vol. 23(2):pp. 47–65, 2009.

[Wes12] Mathias Weske. *Business process management: Concepts, languages, architectures*. Springer Science & Business Media, 2012.