Context-Aware Staged Configuration of Process Variants@Runtime

Aitor Murguzur[1], Xabier De Carlos[1], Salvador Trujillo[1], and Goiuria Sagardui[2]

[1] Software Production Area, IK4-Ikerlan Research Center, Spain
{amurguzur,xdecarlos,strujillo}@ikerlan.es
[2] Embedded Systems Group, Mondragon University, Spain
gsagardui@mondragon.edu

Abstract. Process-based context-aware applications are increasingly becoming more complex and dynamic. Besides the large sets of process variants to be managed in such dynamic systems, 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 runtime where process variants must be customized and executed based on a recognized context. However, there exists a lack of deferral of the entire process variant configuration and execution to perform an automated decision of subsequent variation points at runtime. In this paper, we present a holistic methodology to automatically resolve process variability at runtime. The proposed solution performs a staged configuration considering static and dynamic context data to accomplish effective decision making. We demonstrate our approach by exemplifying a storage operation process in a smart logistics scenario. Our evaluation demonstrates the performance and scalability results of our methodology.

Keywords: Runtime Variability, Late Selection, Context-awareness, Dynamic Software Product Lines, Smart Logistics.

1 Introduction

In recent years, emerging technologies, such as Machine-to-Machine (M2M) communications, Cloud Computing, Service-oriented Computing (SOC), Business Process Management (BPM) and Big Data analytics, have been leveraged to support businesses achieving their goals under changed marked conditions, e.g., reducing the time to market and costs. From the marriage of the latter technologies, smart services are commonly deployed and extended into a variety of smart devices and elastic cloud platforms, to improve decision making, rapid provisioning and deployment, and to provide greater flexibility. Atop of such service-based platforms, process-intensive and event-based applications offer a large number of processes as a catalyst for collaboration, integration and control, e.g., enabling the Business Process as a Service (BPaaS) concept [1].

Variability management for such processes is indeed becoming challenging, e.g., in smart logistics [2]. In this light, new techniques and tools are being developed to address the shortcomings of standard solutions to deal with large sets of process variants and adequate process variants to meet new requirements, context changes and intrinsic complexity. Process variability approaches [3,4] handle different process variants, which are entirely or partially common to several domain stakeholders and assets to which processes are applied. Process variants share a common part of a core process whereas concrete parts fluctuate from variant to variant. In dynamic conditions, such variants need to be context sensitive with the aim of adequately providing multiple stakeholders/assets configurations and reasoning [5], and thus, drive customization based on context information [6]. Therefore, context-awareness demands innovative solutions that allow process-based applications to change during runtime.

This paradigm shift has imposed the emergence of Dynamic Software Product Lines (DSPLs) [7], which support late variability, i.e., defers product configuration to runtime, exploiting traditional Software Product Line (SPL) concepts. Regarding to the BPM field, a configurable process model is capable of dynamically (re-)binding variation points at runtime, considering context information. Once context data is detected from external sensors or new user requirements, the system decides which alternatives of the configurable process must be activated or deactivated and executes the decision via runtime binding. At a glance, the key properties of DSPLs are: (i) support runtime variability, (ii) are able to handle unexpected and environmental changes, (iii) may change variation points at runtime, (iv) may support context-awareness and self-adaptive properties, and (v) may include an automated decision making.

Some of the above-mentioned properties are partially supported by existing work in the literature, such as the Worklets approach [8] which enables user-driven selection of self-contained sub-processes aligned to each activity depending on the particular context at runtime, or the DyBPEL engine [9] which has the ability to adapt running and compliance Business Process Execution Language (BPEL) instances when the corresponding process variant schema evolves. However, there is a lack of deferral of the entire process variant customization and execution to perform an automated decision making at runtime. This would reduce the time to change from one process variant to another, as well as scaling-up and modifying process variant alternatives during operation.

In this paper, we aim to solve such problem by focusing on process models variability at runtime. The main contributions provided by this paper can be summarized as follows: **c1** - we propose a novel methodology and a prototype toolkit called `LateVa` to automatically resolve process variability at runtime; and **c2** - we demonstrate through an experimental smart logistics case study that our methodology is scalable and can be used in dynamic settings.

The paper is organized as follows: In Section 2, we present a detailed overview of the case study at the ACME Corp. and the problems associated with the complexity of its operational processes. In Section 3, we present a methodology based on an automated customization of process variants using the late selection of fragments at runtime to address the issues raised by the case study. The results of our evaluation are detailed in Section 4. Section 5 discusses related work, while we conclude the paper with a summary and future next steps in Section 6.

2 Case Study: Smart Logistics

ACME Corp. is a leading innovative company that provides goods handling systems with a multiplicity of solutions for automated warehouses and storage, baggage handling, sorting systems and picking facilities. ACME Corp. designs and develops Warehouse Management Systems (WMS) for each customer (i.e. offering an ad-hoc solution for each customer to satisfy their business needs) in four areas, namely healthcare, retail, industrial components and food.

In essence, a WMS is a key part of the supply chain which primarily aims to control the movement and storage of goods (also referred to as articles) within a warehouse. Each WMS is complex and comprises a large number of operational processes [2], such as storage, retrieval and picking.

- *Storage*: The goods storage process allows goods to be stored based on different location search strategies (e.g. manual location, fixed location, next empty location, storage unit type, etc.) detailed in a WMS.
- *Retrieval*: The retrieval process enables the complete removal of goods from a warehouse, following disparate extraction strategies (e.g. FIFO, LIFO, least quantity, expiration date, etc.). Such extracted goods are typically moved to an intermediate area for custom shipping configurations.
- *Picking*: The picking process merges both retrieval and storage processes. It consists of taking and collecting goods in a specified quantity before shipment to satisfy customer orders. After each picking operation, the transport unit (e.g. pallet, box) with its remaining articles is routed back to a specific warehouse location based on pre-established storage strategies.

Inside an automated warehouse, the aforementioned operational processes can be modeled and executed by a BPM platform in order to track and control all warehouse flows, and enable multi-agent interaction, such as physical devices (e.g. conveyor systems, transelevators, pick to light systems, RFID, presence sensors, etc.) and warehouse operators (e.g. maintenance manager, workstation agents, picking operator, etc.) [10]. Hence, logistics process automation provides a smart visualization of existing operational processes for each WMS solution and complex event triggering from dozens of sensors; however, adopting a standard BPM platform in a smart logistics scenario presents four major issues:

Large set of process variants. Processes may have common parts and details that can vary for each WMS solution, influenced in various ways by warehouse types, storage areas, location types and conformance checking. As a result, designing ad-hoc processes for each WMS becomes time, resource and cost consuming, as well as an error prone task.

Constantly changing context data. In each automated warehouse, installed sensors are able to provide near real-time data, for instance, about warehouse locations, conveyor systems and the status of goods in transit. It is therefore essential that such events are picked up just-in-time for appropriate decision making. Further, although processes may include different wait states for event catching, inadequate event processing could have a negative impact on the execution of subsequent activities, often requiring manual intervention.

Scalability. An initial warehouse layout can be enlarged to accommodate large amount of orders and/or improve its productivity rates. Previously deployed processes have to be re-designed tackling new requirements. This often involves designing, testing and deploying updated operating processes.

High-availability. WMS execution may not be interrupted and 24/7 service availability is most required. However, any unexpected process execution error would completely stop the system. This would result in expensive system downtime until a system engineer could take corrective action.

3 Process Variants@Runtime

In this section, we present a *fragment-based re-use* methodology used to manage the variability of process models at runtime, that covers the modeling and execution phases of the process life-cycle. In Section 3.1, we present a brief summary of our foundations [11]. Due to space constraints, the formal definitions of `LateVa` foundations are provided as supplement[1]. We detail the different steps of our methodology in Section 3.2. In Section 3.3, we describe the implementation of the methodology in our `LateVa` toolkit.

3.1 Foundations

We follow the Base-Variation-Resolution (BVR) modeling approach [12] from the Software Product Line Engineering (SPLE), which states the separation of model commonality, variability and possible configurations into separate models.

Base Model. The first input of our methodology is a *base model*. It represents the commonality shared by a process family in a particular domain and placeholder activities (variation points) that are subject to vary. This configurable process model may be seen as the intersection or Greatest Common Denominator (GCD) of all related process variants. Variation points identify specific parts in a base model where variant binding occurs. In light of this binding time, we distinguish three types of variation points, as illustrated in Fig. 1: *a*) *static variation point* - resolved at configuration-time (design-time); *b*) *partial variation point* - partially resolved at configuration-time, simplifying the spectrum of runtime fragment choices within a variation model; and *c*) *dynamic variation point* - fully determined at runtime. The latter type of variation point may have two different behaviors: *with-flag* which indicates just-in-time resolution and with *no-flag* in which the resolution is performed at base model instance initialization. In this paper, as will be shown later, we only focus on those two.

Process Fragment. A process *fragment*, or simply fragment, describes a single variant realization option for each variation point within a particular base model. Likewise in a base model specification, it may include different kind of variation points for upholding nested variation points and control flow elements.

[1] `http://tinyurl.com/lateva-formaldef`

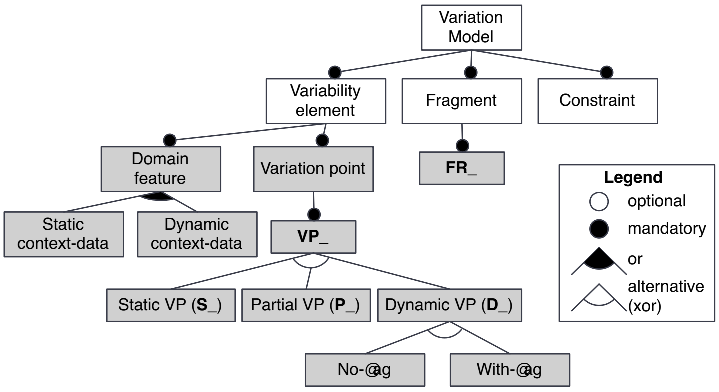


Fig. 1. Variations of the variation model

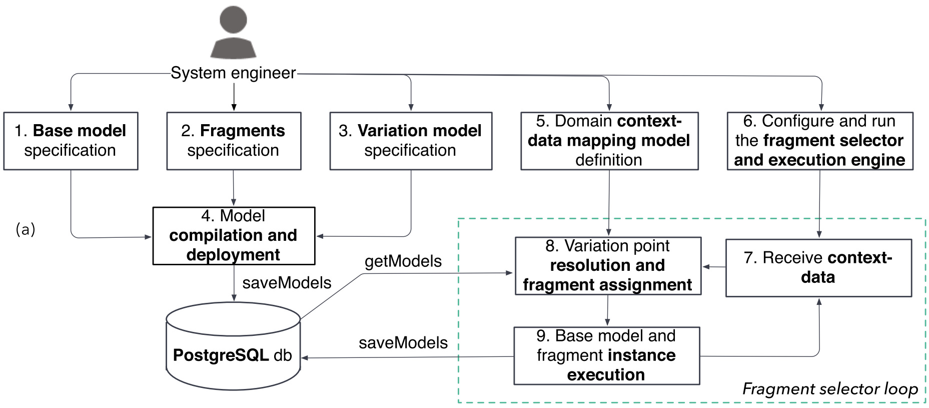
Variation Model. The third input of our methodology is a *variation model* which details all the particularities of a process variant that must be satisfied for valid process variant configuration. It offers abstraction for the base model and its variation points when enhancing a customization, in addition to decision support. Fig. 1 contains a feature diagram for the variation model subtypes that are differentiated in our approach, using the notation proposed by Batory [13]: *a*) *Variability elements* - stand for a family of process variants, their corresponding domain features and variation points. A feature captures a property of the domain that is relevant for a user. It is related to its parent as mandatory, optional, alternative (xor) and or relations [14]; *b*) *Fragments* - characterize variation points' realization options; and *c*) *Constraints* - represent constraints to valid process variant resolutions (complex feature-feature, feature-variable and variable-variable cross-tree relationships).

3.2 Methodology

We describe the methodology in two phases where each phase subsumes several sub-steps. An illustrative overview of the methodology is given in Fig. 2 (a).

Process Variability Specification and Deployment (`LateVa` Modeler). This is achieved via steps numbered 1-5 in Fig.2 (a). Following process family identification, a system engineer provides three inputs as described in Section 3.1: *a*) a base model representing the commonality and variability of a process family (will be exposed as a business service) by employing the OMG standard Business Process Model And Notation (BPMN) version 2.0; *b*) process fragments in BPMN2; and *c*) a variation model for decision support exposed as a feature model given their common industrial adoption [15]. Variation points are modeled using custom BPMN2 activity constructs (e.g. for each type of variation point) supported in our Activiti plugin (see Fig. 3).

(a)



System engineer
1. Base model specification
2. Fragments specification
3. Variation model specification
5. Domain context-data mapping model definition
6. Configure and run the fragment selector and execution engine
4. Model compilation and deployment
saveModels
getModels
8. Variation point resolution and fragment assignment
7. Receive context-data
9. Base model and fragment instance execution
saveModels
PostgreSQL db
Fragment selector loop

(b)

contextVariableName	featureName	contextValue	defaultValue	parseType
operationalFlow	OperationalFlow	HighRates	HighRates	Static
scanner	VP_Scanner	Barcode	Barcode	Static
checkpoint	VP_Checkpoint	P2	P2	Static
boxWidth	boxWidth	280	287	Dynamic
boxLength	boxLength	490	491	Dynamic
boxHeight	boxHeight	501	530	Dynamic
boxWeight	boxWeight	5	6	Dynamic
boxInCorridor	boxInCorridor	2	3	Dynamic

Fig. 2. (a) Methodology (b) Context-data/feature mapping example

In Step 3, the variations for portraying a single variation model (shown in grey in Fig. 1) are: concrete domain feature names and variation points representing domain variability, features with the VP_ extension mapped to variation points in a base model, for fragments we use features with FR_ extension and cross-tree constraints between variability elements and fragments. We distinguish between two context data types, which can be directly mapped to domain features: *static context data* variable/value pairs for static preferences which rarely change over time and are known at base model initialization, and *dynamic context data* variable/value pairs which change and alter over time as they become unpredictable in practice. Therefore, our model could contain features capturing domain abstraction and features associated with domain data.

After all models (base models, fragments and variation models) are defined, they are compiled and deployed to the *models repository*. In this step, we make use of a variation model compiler (the Clafer compiler [16]) to check that predefined features and constraints are well-defined prior to deployment. Since features related to variation points and fragments use direct naming compounds for corresponding process model IDs (see patterns below for creating variation point and fragment features), 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 fragment engine which directly retrieves mapping names and threats exceptions in case of mismatch.

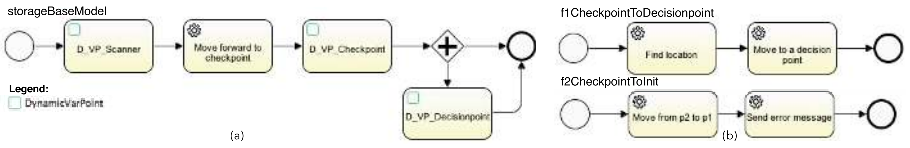


Fig. 3. (a) Storage base model and `D_VP` activities (b) `f1` and `f2` fragment samples

Pattern for representing variation point features

```
{S_,P_,D_} + VP_ + {baseModelVPName} + {parentFeatureName}
```

Pattern for representing fragment features

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

The resolution of process variants is context sensitive. Such resolution could differ from context to context depending on domain context data values and variation model constraints. Those relationships between features, variables, and context data variable/value pairs are represented by *context model mapping* in Step 5 (see Fig. 2 (b) mapping example for the smart logistics case study). Each context data row must define its *contextVariableName* - a concrete variable name in a domain context model, *featureName* - represents a feature or a variable in a variation model, *contextValue* - context information of a context variable, *defaultValue* - a valid value which can be assigned for a context variable, and *parseType* - represents the aforementioned static and dynamic data types.

Staged Process Variants Configuration and Execution (`LateVa` Engine). Base models generated in the previous steps need to be resolved prior to execution. After Step 6, the engine takes system properties for granted. We can configure the engine to work in two different running modes:

– *allin* mode (initialization/startup runtime resolution strategy): runs a two-staged resolution prior to base model instance initialization. Useful when context data values are not altered rapidly over time, so the engine does not necessarily postpone decision making, i.e., variation point resolution and fragment assignment.
– *staged* mode (n-staged online/pure-runtime resolution strategy): performs n-staged online resolution until all variation points are self-determined. Required when a variation point's execution is dependant on fluctuating data (just-in-time dynamic data), and thus faces a critical decision. Critical dynamic variation points are indicated with-flag while non-critical (with no-flag) are resolved at base model instance initialization.

Both running modes start by handling messages, as represented in Step 7 of Fig. 2. When a JavaScript Object Notation (JSON) message is received an internal context object is created based on context model mapping. In addition to static and dynamic variable-value pairs, a JSON message includes three common values, namely *mappingId* - refers to a valid context model mapping identifier,

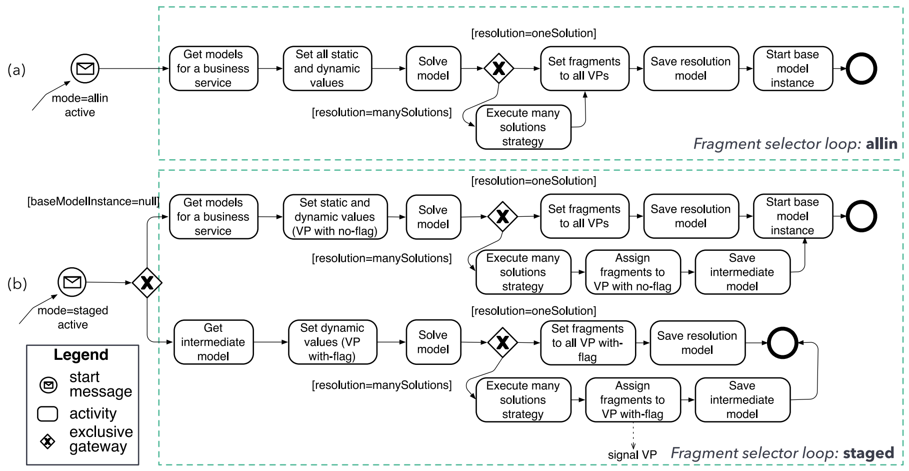

(a)
mode=allin active
Get models for a business service
Set all static and dynamic values
Solve model
[resolution=oneSolution]
Set fragments to all VPs
Save resolution model
Start base model instance
[resolution=manySolutions]
Execute many solutions strategy
Fragment selector loop: allin
(b)
mode=staged active
[baseModelInstance=null]
Get models for a business service
Set static and dynamic values (VP with no-flag)
Solve model
[resolution=oneSolution]
Set fragments to all VPs
Save resolution model
Start base model instance
[resolution=manySolutions]
Execute many solutions strategy
Assign fragments to VP with no-flag
Save intermediate model
Get intermediate model
Set dynamic values (VP with-flag)
Solve model
[resolution=oneSolution]
Set fragments to all VP with-flag
Save resolution model
[resolution=manySolutions]
Execute many solutions strategy
Assign fragments to VP with-flag
Save intermediate model
signal VP
Fragment selector loop: staged
Legend
start message
activity
exclusive gateway

Fig. 4. (a) *allIn* and (b) *staged* running modes of the fragment selector engine

service - describes a business service (base model) that should be executed, and *instance* - identifies a running base model instance. It could also contain extra data that will be ignored by the engine, but recognized by others (e.g. data visualization). If context model mapping and business service description are encountered, the engine follows a pre-established run strategy.

Example of a JSON message for the smart logistics case study

```
{message:{"date":"2013-10-25", "time":"09:40:13", "mappingId"
    :"Geneva", "service":"StorageOP", "instance":"4401", "
    domain":"SmartLogistics", "operationalFlow":"HighRates",
    "scanner":"Barcode", "checkpoint":"P2", "boxWidth":"280",
     "boxLenght":"480", "boxHeight":"520", "boxWeight":"5", "
    boxInCorridor":"0"}}
```

Bearing in mind the two running strategies, steps 8-9 in Fig. 2 are executed differently for each configuration (see Fig. 4). In *allin* mode, the fragment selector loop starts by retrieving base model and variation model definitions for a given business service. Here, we automatically transform a variation model to a constraint satisfaction problem (CSP). Proposals using constraint solving to reason on feature modes have been studied over the past two decades [17]. In our case, every optional feature becomes a boolean variable of the CSP with domain {`false`,`true`} or {0,1}, whereas every mandatory domain feature becomes a {`true`} or {1} variable.

The second activity in Fig. 4 (a) determines context data values in two-stages. Firstly, static context data is parsed for putting features into one:

Example of static feature constraint

```
vSpecModel.addConstraint(one(concreteFeature));
```

Additionally, dynamic values are specified by setting `Integer` constraints:

Example of dynamic feature constraint

```
concreteFeature.addConstraint(equal(joinRef($this()),
    constant((int) constraintValue)));
```

After parsing all context values and setting up constraints to the variation model, the solver starts running propagation and search to derive possible valid configurations. We handle three situations:

- *No solution* after allin: Apart from the functionality provided in Fig. 4 (a), the fragment selector engine must deal with unexpected situations, such as no solution found after model solving. Pro tem, the engine rollbacks all operations for an unexpected message.
- *One solution* after allin: The engine assigns fragments to applicable variation points from the resulting solution, saves this model (resolution model) in the models repository and enhances a base model instance. When the execution reaches a variation point, it starts a valid fragment execution.
- *Many solutions* after allin: When the solver returns more than one valid solution for the current context, we may select between four different strategies: (i) *get-first* to get the first feasible fragment, (ii) *get-default* to select the fragment marked as default, (iii) *recommender* system to decide which is the suitable solution for the given context, and (iv) *manual-selection* to enable system-user decision making. In this paper, the engine automatically runs a *get-first* strategy, i.e., it returns the first correct resolution from all possible configurations and establishes this as an input for the "one solution" flow.

The second run mode, namely the *staged* mode, may activate two different branches depending on whether the received context data is for a running instance or not. This is indicated by the *instance* variable.

In a first stage (the upper workflow in Fig. 4 (b)), both static and dynamic constraints for dynamic variation points with no-flag are set (with-flag indicates that a variation point is critical and thus needs just-in-time data for its proper resolution). In the event of many solutions, the get-first strategy is performed to assign fragments for dynamic variation points with no-flag and an intermediate model is saved. Otherwise, one solution flow performs assignations for all dynamic variation points in order to ensure a valid customization.

The n-staged flow (the lower workflow in Fig. 4 (b)) is activated to handle "just-in-time" context messages. For each dynamic variation point in a wait state (with staged flag = `true`), the engine establishes constraints for an intermediate model which is restored from the models repository by the specified instance ID. The solver uses this altered model as an input, and concludes if a sound configuration exists. If just one exists, all with-flag dynamic variation points are determined by a suitable fragment. Many solutions after n-staged, however, implies the activation of a many-solution strategy (e.g. get-first) to assign a fragment for pending dynamic variation points with-flag. These are then reactivated by signal-catching to initiate a concrete fragment instance.

3.3 Implementation in LateVa Toolkit

We implement our methodology in the `LateVa` toolkit to manage the variability of process models at runtime. The implementation is standalone and employs open source frameworks such as Activiti[2] for base model and process fragment modeling and execution, Clafer[3] for variability representation with its alternative backend using Choco[4] constraint solver, ActiveMQ[5] as the engine broker and Camel[6] for integration. A prototype implementation of the toolkit encompasses two modules: the `LateVa-modeler` for representing models (as an Activiti extension) and the `LateVa-engine` for executing late variability.

4 Evaluation

In this section, we perform an experiment to synthesize a storage process for the ACME Corp. case study and discuss the experimental results and threats to validity of the proposed methodology.

4.1 Experimental Setup

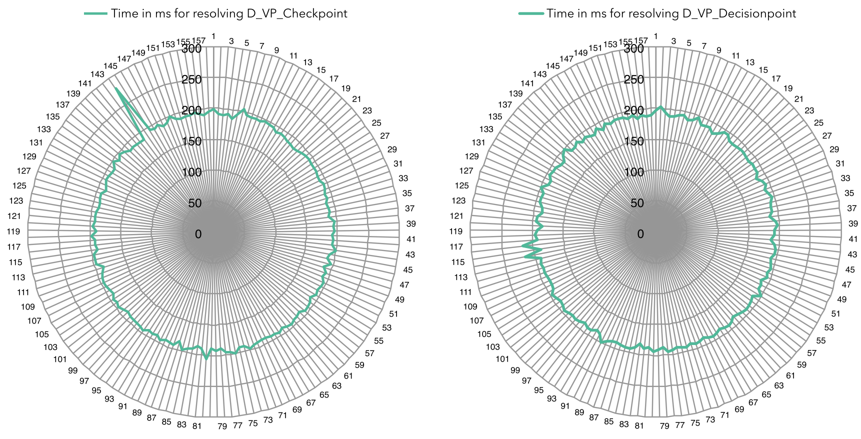
We developed a case scenario to automatically configure and enhance a storage process to test an automated warehouse operation in the Geneva apparel industry. The warehouse layout consists of a single material entry point in which goods are packed in carton boxes (288x492x531mm), two corridors each containing 2000 locations, 2 picking workstations and a single material retrieval point. Due to space limitations, the full description of the storage process example is given online at: `http://aitormurguzur.com/projects/lateva/caise2014`

4.2 Results of Discussion

We generated 1 base model with 3 dynamic variation points (1 with no-flag: `D_VP_Scanner`, and 2 with-flag: `D_VP_Checkpoint` and `D_VP_Decisionpoint`, as depicted by Fig. 3), 5 fragments and 1 variation model with 22 features and 8 constraints, covering 3x2x2 = 12 process variant customizations for the Geneva layout. These models were used to generate **a test case for 157 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.

In this paper, we focus on understanding the performance of our approach in relation to the smart logistics case study. We use the non-intrusive `perf4j` library to perform measurements. Dynamic variation point execution performance is measured by total processing time. Fig. 5 shows the minimum, maximum and

[2] `http://activiti.org`
[3] `http://clafer.org`
[4] `http://www.emn.fr/z-info/choco-solver`
[5] `http://activemq.apache.org`
[6] `http://camel.apache.org`



min: 184 ms | **max**: 281 ms | **avg:** 193.0318 ms | **diff**: 97 ms

min: 182 ms | **max**: 216 ms | **avg:** 191.4076 ms | **diff**: 34 ms

Fig. 5. TTR of dynamic variation points with-flag

average time taken by each with-flag dynamic variation point before fragment execution. Consequently, the average time-to-resolution (TTR) that a base model instance has to wait for using this kind of variation points is **192.2197 ms**. The delay can be omitted by including events and exclusive gateways in the modeling phase, however, `LateVa` solves several issues raised at the beginning of the paper:

Large set of process variants. This is achieved by separating operational process variability in disjointed models. The key model (the variation model) captures variability at the domain level in which different processes have been assembled. This allows users to focus on domain concepts rather than on each BPMN2 elements.

Constantly changing context data. The variation model may contain two types of domain features: features for domain abstraction and features/variables mapped to context data. Dynamic context data can be controlled by establishing constraints and automatically customizing process variants.

Scalability. Our approach supports *variability by extension* [18], i.e., it allows variable segments of operational processes to be captured in fragments and weaved into a base model. Consequently, the base model is minimal in that it only contains elements common to all processes. Specific variation points elements (i.e. fragments) are added and/or updated when needed. This also applies to new context data types.

High-availability. Unexpected situations are dealt effectively and appropriate fragments are updated when necessary. Fragment replacement only affects new configurations while running base model instances will assure their correct execution path.

Looking beyond the particulars of the case study, we lend to generalization by checking how our approach operates with increasing `LateVa` fragments, i.e., approach scalability. We added 50 extra fragments to `D_VP_Checkpoint`, so that

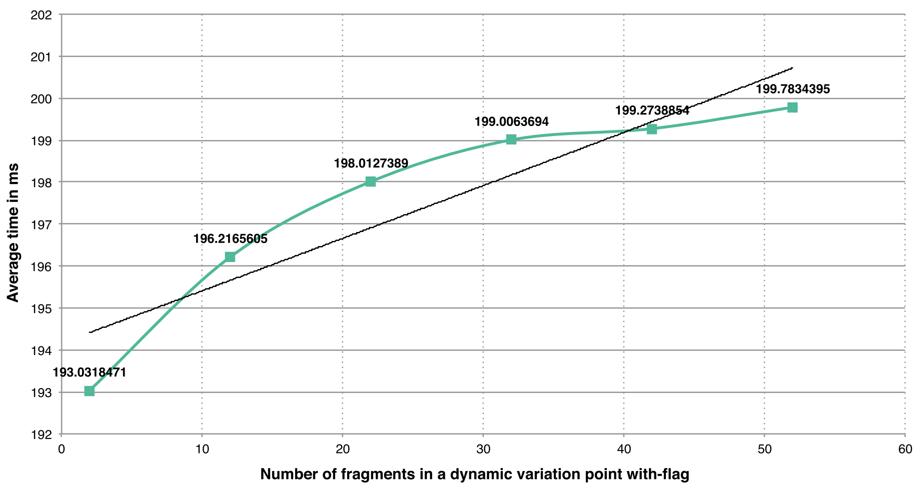


Fig. 6. TTR of `D_VP_Checkpoint` by increasing fragments in the variation model

the number of possible variant configurations increases to 312 (3x52x2). First results are encouraging which indicate that the engine suffers a minimal delay of **1.3503 ms** when the number of applicable fragments grow for a particular dynamic variation point with-flag, as illustrated in Fig. 6. All datasets are available at: `http://git.io/3sUrCA`

4.3 Threats to Validity

Our experiments did not consider a certain number of factors that may affect our proposed methodology and generalized outcomes on scalability.

- *Correctness*: Our approach guarantees the syntactical correctness (correct structure of process variants), but not behavioral (soundness [19], e.g., by avoiding deadlocks and livelocks) correctness of the resulting process variants. For the former, we avoided disconnected nodes by representing process fragments with initial and final states and enabling "call activity" behavior from BPMN2 elements within new activity constructs (variation points). However, it may become impossible to guarantee behavioral correctness of the customized models adopting the variability by extension strategy [18]. In our case, this is due to the large number of possible combinations of solver options, considering both static and dynamic context variable/value pairs at runtime, as well as evolutionary variation models, i.e., the process engineer may also add or modify fragments in the variation model.
- *Dynamic context data*: In this paper, we only employed `Integer` values for constraint solving, but the variation model should consider other data types, such as, strings and reals in future releases.
- *Base model, fragment and variation model scalability*: Although initial tests on model scalability show promising results, we plan to consider more scalability and performance tests, and also evaluate the presented methodology

against different industrial case studies. In addition, only dynamic variation points were considered in the presented case study; however, static and partial variation points need to be incorporated in future tests.

5 Related Work

The work presented is related to other fields of research, such as process variability, context-aware configuration of process variants, process flexibility and DSPLs. In this section, we briefly introduce related research in these areas and explain the novelty of our proposed approach.

Previous work on *process variability* [18,20] can be divided into two main groups: approaches adopting a single configurable process model [3,4,21] and fragment-based re-use approaches [6,9,8]. C-EPC [3] is an extension of the Event-driven Process Chain (EPC) to support multiple process families of EPC process variants by means of configurable nodes and configurable alternatives. Orthogonally, the PESOA project [4] was proposed to provide mechanisms to capture variability of UML activity diagrams and BPMN, enriching them with stereotype annotations using feature models in an abstract way (they do not transform variation points to selected variants). C-YAWL [21] is an extension of the Yet Another Workflow Language (YAWL) which applies hiding/blocking operations to customize a configurable process model. They all provide useful methods to deal with process variability; however, they are merely focused on design-time process variability, rather than customizing process variants at runtime. Furthermore, they are based on conceptual modeling rather than executable process models.

Context-aware process modeling and variability has been also studied in the past. A context-aware framework for the explicit modeling of context is provided in [22], which includes system stakeholders' reasoning about business processes in appropriate manner. The importance of context sensitive process modeling has been also discussed in other studies [5], in order to deal with changing user requirements and the complex and dynamic nature of environments. In a similar vein, context-awareness has been translated to the process variability field. For instance, the Provop approach [6] provides five steps to customize base models based on context information. Although the latter allows for context-aware process variants customizations employing a base model, such customizations are user-supported rather than automated and they are not executable in practice.

Process variability and *flexibility* concepts are closely connected. Process flexibility is concerned not only with variability but also with runtime aspects of process models, as described by Reichert and Weber [23]. Hence, runtime variability may be seen as a flexibility type (also referred to as late selection or late binding) which defers placeholder activity resolution to runtime. The work that is closer to our proposal is the Worklets approach [8], which enables a dynamic runtime selection of self-contained sub-processes aligned to each activity depending on the context of the particular instance. When activities become enabled appropriate fragment selection is achieved by using Ripple Down Rules (RDR), which include hierarchically organized selection rules. With respect to the mentioned work, the main novelties of our approach are two-fold: we provide a staged configuration and execution of process variability at runtime, which can perform customizations in different ways (allin, staged). Secondly, all customizations are performed automatically while in Worklets the selection is realized interactively by end-users (user interaction).

Context-awareness and intrinsic complexity of environments has caused *DSPLs* to support late variability in systems in order to cater for changes at runtime. Baresi et al. [9] propose a methodology based on Common Variability Language (CVL) and DyBPEL for managing process reconfiguration at runtime using DSPLs. This proposal has been focused on handling process variants at design-time and supporting adaptation at runtime; however, our approach merely binds variation points at runtime. Despite the fact that DSPLs have been applied to other research areas such as, model-driven engineering [24] or service-oriented systems [25], to the best of our knowledge this is the first work tackling the issue of DSPLs in process variability.

6 Conclusion

In this paper, we have addressed the problem of a smart logistics scenario and presented a novel fragment-based re-use approach for an automated management of process models variability at runtime. We introduced a methodology implemented in the `LateVa` toolkit, to manage large sets of variants that utilize context data and constraint solving for process variant customization. In the experiment, we developed a base model, 5 fragments and a variation model for a warehouse storage operation. The evaluation generated 157 tests and concluded that the approach provides an average TTR of **192.2197 ms** for dynamic variation point with-flag. The initial scalability tests for our approach have provided positive results for its adoption as an alternative in scenarios where: large set of process variants, dynamic context data, scalability and/or high-availability are the norm, not the exception. Our future work will involve a large-scale empirical evaluation using different real case studies while carefully collecting scalability/performance metrics and dealing with models complexity and maintainability. We also plan to concentrate on three parts: testing of other many-solution strategies when the solver gets many solutions, abstractions for enabling user-defined dynamic process configuration [26], and how to deal with data variability for the automated generation of context model mappings.

References

1. Böhmer, M., Daniluk, D., Schmidt, M., Gsell, H.: Business object model for realization of individual business processes in the logistics domain. In: Efficiency and Logistics, pp. 237–244 (2013)
2. Derguech, W., Gao, F., Bhiri, S.: Configurable process models for logistics case study for customs clearance processes. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part II. LNBIP, vol. 100, pp. 119–130. Springer, Heidelberg (2012)
3. Rosemann, M., van der Aalst, W.: A configurable reference modelling language. Information Systems 32(1), 1–23 (2007)

4. Puhlmann, F., Schnieders, A., Weiland, J., Weske, M.: Variability Mechanisms for Process Models. Technical report (2005)
5. Saidani, O., Nurcan, S.: Context-awareness for adequate business process modelling. In: RCIS, pp. 177–186 (2009)
6. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: The provop approach. Journal of Software Maintenance and Evolution: Research and Practice 22(6-7), 519–546 (2010)
7. Bencomo, N., Hallsteinsen, S., de Almeida, E.S.: A view of the dynamic software product line landscape. Computer 45(10), 36–41 (2012)
8. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
9. Baresi, L., Guinea, S., Pasquale, L.: Service-oriented dynamic software product lines. Computer 45(10), 42–48 (2012)
10. Ao, Y., He, W., Xiao, X., Lee, E.: A business process management approach for rfid enabled supply chain management. In: ETFA, pp. 1–7 (2010)
11. Murguzur, A., Sagardui, G., Intxausti, K., Trujillo, S.: Process variability through automated late selection of fragments. In: Franch, X., Soffer, P. (eds.) CAiSE Workshops 2013. LNBIP, vol. 148, pp. 371–385. Springer, Heidelberg (2013)
12. Bayer, J., Gerard, S., Haugen, Y., Mansell, J., Müller-Pedersen, B., Oldevik, J., Tessier, P., Thibault, J.P., Widen, T.: Consolidated product line variability modeling. In: SPLC, pp. 195–241 (2006)
13. Batory, D.: Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
14. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report (1990)
15. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: VaMoS (2013)
16. Antkiewicz, M., Bąk, K., Murashkin, A., Olaechea, R., Liang, J., Czarnecki, K.: Clafer tools for product line engineering. In: SPLC, Tokyo, Japan (2013)
17. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
18. Rosa, M.L., van der Aalst, W.M., Dumas, M., Milani, F.P.: Business process variability modeling: A survey. ACM Computing Surveys (2013)
19. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: Classification, decidability, and analysis. Form. Asp. Comput. 23(3), 333–363 (2011)
20. Valença, G., Alves, C., Alves, V., Niu, N.: A Systematic Mapping Study on Business Process Variability. IJCSIT 5(1) (2013)
21. Gottschalk, F., van der Aalst, W.M., Jansen-Vullers, M.H., Rosa, M.L.: Configurable workflow models. IJCIS 17(2) (2008)
22. Balabko, P., Wegmann, A.: Context based reasoning in business process models. In: IRI, pp. 120–128 (2003)
23. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer (2012)
24. Morin, B., Barais, O., Jezequel, J., Fleurey, F., Solberg, A.: Models@ run.time to support dynamic adaptation. Computer 42(10), 44–51 (2009)
25. Parra, C., Blanc, X., Duchien, L.: Context Awareness for Dynamic Service-Oriented Product Lines. In: SPLC, pp. 131–140 (2009)
26. Lapouchnian, A., Yu, Y., Mylopoulos, J.: Requirements-driven design and configuration management of business processes. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 246–261. Springer, Heidelberg (2007)