Context variability modeling for runtime configuration of service-based dynamic software product lines

Conference Paper · September 2014

DOI: 10.1145/2647908.2655957

Aitor Murguzur
Ikerlan
13 PUBLICATIONS 42 CITATIONS
SEE PROFILE

Rafael Capilla
King Juan Carlos University
109 PUBLICATIONS 1,092 CITATIONS
SEE PROFILE

Salvador Trujillo
Ikerlan
67 PUBLICATIONS 723 CITATIONS
SEE PROFILE

Óscar Ortiz
Universidad Politécnica de Madrid
7 PUBLICATIONS 46 CITATIONS
SEE PROFILE

Context Variability Modeling for Runtime Configuration of Service-based Dynamic Software Product Lines

Aitor Murguzur
IK4-Ikerlan Research Center
Arrasate, Spain
amurguzur@ikerlan.es

Rafael Capilla
Rey Juan Carlos University
Madrid, Spain
rafael.capilla@urjc.es

Salvador Trujillo
IK4-Ikerlan Research Center
Arrasate, Spain
strujillo@ikerlan.es

Óscar Ortiz
Technical University of Madrid
Madrid, Spain
oscar.ortiz@upm.es

Roberto E. Lopez-Herrejon
Johannes Kepler University
Linz, Austria
roberto.lopez@jku.at

ABSTRACT

In emerging domains such as Cloud-based Industrial Control Systems (ICSs) and SCADA systems where data-intensive and high performance computing are needed, a higher degree of flexibility is being demanded to meet new stakeholder requirements, context changes and intrinsic complexity. In this light, Dynamic Software Product Lines (DSPLs) provide a way to build self-managing systems exploiting traditional product line engineering concepts at runtime. Although context-awareness is widely perceived to be a first-class concern in such runtime variability mechanisms, existing approaches do not provide the necessary level of formalization to model and enact context variability for DSPLs. This is crucial for operational analytics processes since variant configuration could differ from context to context depending on diverse data values linked to context features and cross-tree constraints in a feature model. In this paper, we propose a context variability modeling approach, demonstrate its applicability and usability via a wind farm use case, and present the fundamental building blocks of a framework for enabling context variability in service-based DSPLs which provide Workflow as a Service (WFaaS).

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.2 [**Design Tools and Techniques**]: Miscellaneous

General Terms

Design

Keywords

Context Variability, Context Awareness, Data-aware Systems, Process Variability

SPLC '14, September 15 - 19 2014, Florence, Italy
Copyright 2014 ACM 978-1-4503-2739-8/14/09 ...$15.00.
http://dx.doi.org/10.1145/2647908.2655957

1. INTRODUCTION

Data-intensive and process-based distributed applications, such as in Cloud-based Industrial Control Systems (ICSs) and SCADA systems, demand highly configurable and adaptable systems to meet new stakeholders requirements, context awareness and intrinsic complexity. Atop of such ICSs infrastructures (e.g. offshore windmills), context-aware applications may offer a large number of Operation and Maintenance (O&M) analytics processes as a catalyst for collaboration, integration and control between inherent stakeholders and software services, as well as data analysis and decision support. In order to engineer and intervene in those process-aware applications, variability management becomes a primary concern to maximize re-use and exploit commonality and variability of systems to produce higher-quality processes in less time [12]. However, although traditional Software Product Lines (SPLs) have been successfully applied during the last years, data-intensive and context-aware ICSs demand new solutions for adaptive systems families [24].

This paradigm shift has imposed the emergence of Dynamic Software Product Lines (DSPLs) [16, 18] as a cornerstone to defer product configuration and a seamless variability management at runtime. Apart from empowering dynamic variability (also referred to as runtime variability [7]), DSPLs also provide other features to support unexpected and environmental changes, dynamic variation points, self-adaptive properties, and automated decision making. Hence, in data-intensive SPLs (e.g. wind farms) and due to the complex nature of O&M computational models (e.g. MATLAB) in connection with the vast amounts of data and high variability of processes, context awareness plays a crucial role in the adoption of DSPLs [17, 1]. Despite of the fact that DSPLs have been in the focus of existing work in the literature, such as service-based systems [19, 21], mobile systems [13], and self-adaptive systems [10, 2], there is less coverage of approaches dealing with context variability management in service-based DSPLs. This is specially important for context-aware executable analytics processes (exposed as software services[1]), where O&M process variants are automatically customized and executed at runtime in subsequent

[1]Executable analytics processes are provided as services, enabling the Workflow as a Service (WFaaS) concept.

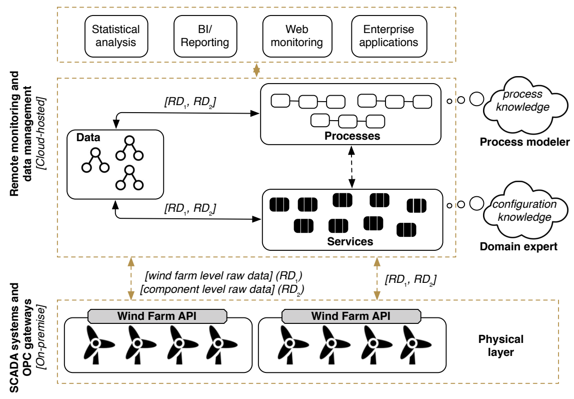


Figure 1: Process-based O&M data analytics.

data interactions (e.g. from wind farms APIs and data services), considering context information (e.g. raw data).

In this paper, we present the fundamental design building blocks of an approach for context variability modeling and execution in service-based DSPLs. Overall, this paper makes the following *contributions (C)*:

- **C1**: We outline the main requirements needed for context variability management in service-based DSPLs by first exemplifying a wind farm case study.
- **C2**: We propose the main design considerations of a framework to customize service-based DSPLs.
- **C3**: We describe a strategy for modeling context variability by employing feature models for variability implementation and ontologies for domain modeling (data and process context), to cope with context data association ambiguity and heterogeneity.

These characteristics will enable the domain and context variability modeling and execution, in order to get the best performance as possible and ensure scalability for rapid and effective large-scale runtime variability management in DSPLs. The rest of the paper is organized as follows: In Section 2, we present an overview of the motivating wind farms scenario and the main issues to be tackled in such variability ecosystem. In Section 3, we analyze the main requirements of our approach. Section 4 presents the overall architecture and details the individual framework design considerations. Section 5 discusses the functionality and usefulness of the presented approach. Related work is presented in Section 6. Finally, we conclude the paper and present the direction of the future work in Section 7.

2. CASE STUDY: O&M FOR WIND FARMS

In Cloud-based remote monitoring and data management (as illustrated by Figure 1), the data needed to actually execute process activities is much broader than the typical process-related data, e.g., as in Internet of Things applications [22]. This is specially required in windmill farms due to the complex nature of computational *services* aligned to subsequent activities of executable analytics *processes*, in connection with the vast amounts of diverse and unstructured

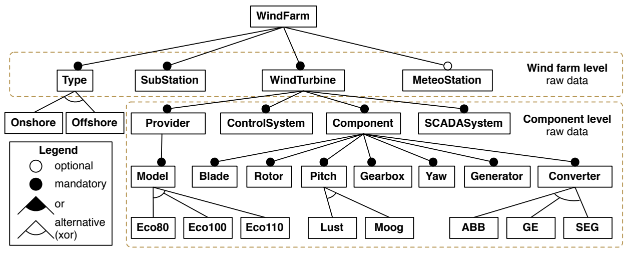


Figure 2: A simplified wind farm ecosystem.

raw *data* provided by sensors and SCADA systems. Such processes are employed for O&M operations of onshore/offshore wind farms (e.g. active power vs wind speed), which produce energy by grouping hundred individuals of wind turbines in an extended area of ground/water and converting kinetic energy from the wind into electrical power. For instance, our industry partner Alstom[2] collects close to 1 Terabyte from all wind farms/day, monitoring 22 variables per turbine/second and using the IEC 61400-25 standard.

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. Hence, at wind farm level, multiple stakeholders (O&M system providers, business owners, etc.) could store, process, analyze, and access real-time and historical data from different OPC gateways of installed SCADA systems, as well as sub-stations and meteo-stations. However, data ambiguity and heterogeneity problems of raw data must be solved before considering data analytics. For instance, *wind farm level raw data* (see Figure 1) could differ from one wind turbine provider to another (e.g. by offering different turbine models Eco80, Eco100, Eco110, etc.), such as Alstom, Siemens, Vestas or other industry players, thus ranging from a variety of data (e.g. logical device name, active power generation, temperature of water at outlet, etc.) that can by taken/interpreted as a time-series feeds (e.g. every 10min).

Beyond the particulars of different wind turbine providers, each onshore/offshore wind turbine consists of a number of components from different Original Equipment Manufacturers (OEMs), which offer hardware that can be assembled to form a convenient equipment. A simplified version is shown in Figure 2, where seven main components are considered: (i) wind turbine `Blades`, (ii) a `Pitch` for controlling blades' 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`. Similarly to high-level data provided by wind farms, such components (each having an individual data model) may also emit fine-grained *component-specific raw data* that can be analyzed 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, just to name a few [26].

In the aforementioned process-based O&M scenario and due

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

to the variety and dynamicity of data from windmill farms, data analytics processes should be configured and executed at runtime in subsequent data interactions. This is especially valid for sensor and machine data (such as in wind farms), which forces us to react on incoming data with real-time processing and thus to defer decisions to runtime, where analytics process variants need to be customized and executed based on recognized context, i.e., context-aware runtime configuration. This would reduce the time to change from one variant to another, as well as scale and modify variant alternatives during enactment. Under this new disposition, the adoption of standard[3] or Cloud-based[4] Workflow Management Systems (WfMSs) presents two major issues:

1. **Context data variety and dynamicity:** As mentioned, each wind turbine and component type can have a differentiated data model. Disparate data variable/value pairs coming from context could refer to the same wind farm data semantics (e.g. rotor speed, logical device name, etc.). We can distinguish between two context data types, which can be directly mapped to context features: *static context data* variable/value pairs for static preferences which rarely change over time and are known prior to analytics process execution (e.g. logical device name), and *dynamic context data* variable/value pairs which change and alter over time as they become unpredictable in practice (e.g. rotor speed). Therefore, we should be able to model features capturing static domain abstraction and features associated with context data, i.e., also referred to as *context data variability*.

2. **Context-aware process variability:** Analytics processes may have common parts and details that can vary for each wind farm, influenced in various ways by wind turbine types, location and component types. As a result, designing ad-hoc processes for each wind farm may become time, resource and cost consuming, as well as an error prone task. In contrast, taking into account context data variability, we should be able to configure and execute analytics processes at runtime by means of automated decision making on suitable fragments for current context data. This would reduce the time to change from one analytics process variant to another for each particular context, as well as ease managing large sets of process variants and enabling a DSPL to support *runtime variability of process models*.

3. RUNTIME REQUIREMENTS FOR MANAGING CONTEXT VARIABILITY IN SERVICE-BASED DSPLS

The aforementioned issues for modeling context aspects in a service-based product line for the wind farm domain are driven by several runtime concerns that demand the use of DSPL solutions. Most of these runtime concerns deal with the way in which context information collected from the wind farm ecosystem is treated in order to achieve the maximum performance of wind generators. In essence, we can list the following reasons that motivate the adoption of a dynamic variability approach for the wind farm product line: (i) *context-awareness* - context information may vary between different brands of generators, (ii) *self-adaptive* - we should be able to handle unexpected and environmental changes (e.g. new context features might be integrated at runtime), (iii) *runtime variability* - context-aware analytics processes and/or tasks can be automatically configured at runtime to provide the kind of results that diverse stakeholders demand, and (iv) *multiple binding* - analytics processes can be bound and/or launched at later binding times.

In terms of multiple binding alternatives, different analytics process variants must be able to configure during the first start-up (i.e. at post-deployment time) and during the normal system operational mode (i.e. at pure runtime). For instance, the former can be adopted when context data values are not altered rapidly over time, so the variability engine does not necessarily postpone decision making, whilst the latter execution is dependent on fluctuating data (just-in-time dynamic data), and thus faces a critical decision. It should also be possible the transition between these two binding modes, even during operation. Consequently, it is crucial to provide means of adaptation support in wind farms O&M platforms which demand runtime variability mechanisms in addition to the representation of context variability related to processes, data and physical environment itself. Hence, we summarize the following *requirements (R)* for handling the contextual and dynamic variability in service-based DSPLs:

- **R1: Model the context variability of wind farm data** - The context information gathered as raw data coming from external sensors, SCADA systems and from other ICS devices (wind turbines' components) must be modeled as context features. We will also need to support context variability related to analytics processes, data variety and physical environment where similar or same context features can be named or positioned differently for certain types of wind farms.

- **R2: Dynamic behavior of context features** - Context features must be activated and deactivated accordingly to varying context conditions. It should be possible to activate a context feature depending on other features (e.g. when one component type requires the inclusion of other components). Moreover, the runtime variability mechanism should support the inclusion or removal of context features at runtime in case of deriving new functionalities from new or existing wind turbine generators. Such changes will affect the structural variability represented in the simplified wind farm feature model of Figure 2.

- **R3: Multiple and dynamic binding** - Runtime configuration and reconfiguration activities of O&M analytics processes, which are executed in subsequent data interaction through determining context variability at runtime, are necessary. Although each variant could be customized in different operational modes (i.e. configuration, deployment-time, start-up, and pure-runtime), the solution for the wind farm DSPL must provide runtime-oriented multiple binding support (i.e. start-up and pure-runtime) for runtime variant resolu-

[3]`http://activiti.org/`
[4]`http://www.effektif.com/`

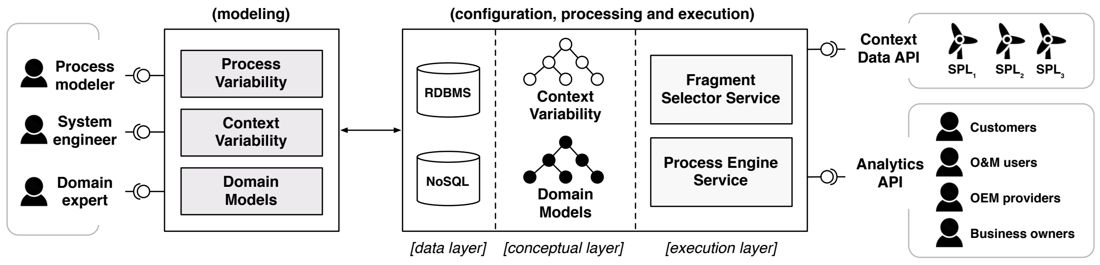


Figure 3: Overview of the framework.

tion and execution. Further, such DSPL requires re-binding mechanisms that can be enacted dynamically to change from one operational mode to another.

In this paper, we focus primarily on the requirement **R1**, i.e., in context and dynamic variability modeling aspects, and present the design building blocks of an approach necessary to satisfy all the aforementioned requirements, enabling dynamic variability for service-based DSPLs.

4. A CONTEXT-AWARE VARIABILITY APPROACH FOR SERVICE-BASED DSPLS

Our approach aims to provide a solution for context variability modeling by means of using different context variability and domain modeling strategies, and an execution framework capable of self-configuring service-based DSPLs based on context information at runtime. This includes process variant and context variability modeling, automatic context data acquisition, processing, and variant configuration for service-based context-aware systems. For that purpose, as illustrated in Figure 3, our framework enables dynamic variability modeling and execution for multiple stakeholders and data realms.

4.1 Overview

In order to satisfy different stakeholders' needs (e.g. time and cost constraints), during the modeling phase the process modeler may first create *base models* and *fragments* to represent context-aware process variability (exposing base models as software services - WFaaS), placing common activities and custom dynamic variation points for data interaction activities and variants (e.g. Service Tasks for invoking available computational models service). These models can be deployed to the models repository (RDBMS) and made directly available to the application through the analytics API. The *context variability model* encodes the applicable alternatives considering context data, available processes and physical environment variability. The formal semantics of feature models, process types (base models and fragments), data endpoints and data classes are provided by an ontology-driven *domain model* description.

During the execution, in each analytics process request the *fragment selector service* searches for available base models that can accomplish the desired service request and make use of the process engine service to start base model execution. The base model instance execution resolves dynamic variation points by collecting data from data realm REST API (context data API) and data services to adequately select an applicable fragment considering context information. Hence, data retrieved from the data realm API is saved in a data-store (NoSQL) and utilized by the fragment selector service to make an automated decision on candidate fragment using context variability and domain models. Analytics API makes our approach bidirectional, i.e., different stakeholders (e.g. customers, O&M users, etc.) can notify on analytics request to the system, and receives alerts and data for visualization, e.g., via WebSockets.

In the following, we focus on the modeling part of Figure 3, in order to meet the **R1**.

4.2 Modeling Dynamic Variability

During the modeling phase, our approach enables separation-of-concerns by employing: base models and process fragments for *process variability* using an executable process modeling language, such as the OMG standard Business Process Model And Notation (BPMN) version 2.0, variability models as feature models for *context variability* modeling, and ontologies and data models for *domain modeling.*

4.2.1 Process Variability

Our framework is based on a fragment-based re-use methodology [12] (also referred to as positive variability [14]) which promotes "design by reuse" by breaking process variability into disjointed models. For that purpose, we differentiate three main inputs to model commonality and variability in disjointed models (shown in gray in Figure 4): (i) base models, (ii) fragments, and (iii) variability models respectively. For a more detailed discussion regarding process variability modeling, we refer to our previous work in [23].

A *base model* represents the commonality shared by an analytics process family and placeholder activities (variation points) that are subjected to vary. In contrast, variation points identify specific parts in a base model where variant binding occurs, i.e., where suitable fragment selection is performed based on current context data. In our view, a base model is formalized as follows: 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 subset of variation points: $\forall\ \mathcal{BMI} \in \mathcal{BM}$, $\mathcal{BMI} = (\{\mathcal{C}\}, \{\mathcal{SVP} \subset \mathcal{VP}\})$.

A process fragment, or *fragment* for short, describes a particular configuration option for each variation point within

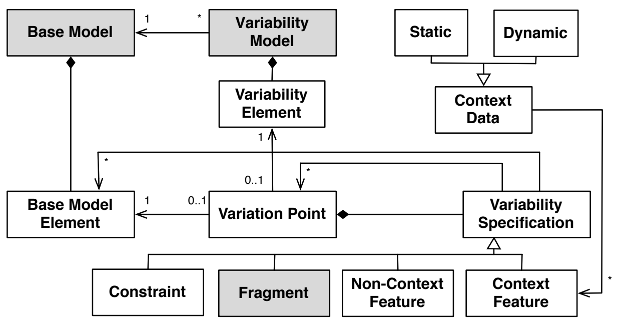


Figure 4: Variability model meta-model.

a base model. In formal terms, a fragment $\mathcal{F}$ is defined as a 2-tuple $<\mathcal{PE}, \mathcal{FVP}>$, being $\mathcal{PE}$ the set of fragment elements and $\mathcal{FVP}$ the set of possible fragment variation points. Every fragment instance $\mathcal{FI}$ derived from $\mathcal{F}$ consists of a subset of fragment elements and a subset of fragment variation points: $\forall\ \mathcal{FI} \in \mathcal{F}, \mathcal{FI} = (\{\mathcal{SPE} \subset \mathcal{PE}\}, \{\mathcal{SFVP} \subset \mathcal{FVP}\})$.

A *variability model* enables the description of all particularities of a process variant that must be satisfied for valid process variant configuration for the current context. Fundamentally, it offers abstraction for a base model and its variation points when enhancing a customization, in addition to decision support. Formally, 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 dynamic variation points, $\mathcal{F}$ the set of possible applicable fragments for each variation point, and $\mathcal{O}$ the constraint sub-model made up of features and variables.

As depicted by Figure 4, for context-aware process variability description a variability model employs a feature model as follows: (i) non-context features, (ii) features that are linked to process variability elements such as dynamic variation points (`D_VP_`) and fragments (`F_`), (iii) context features that are mapped to both static and dynamic context data variable/value pairs, and (iv) cross-tree constraints to represent conditions or restrictions for valid process variant resolutions. For instance, a fragment feature can be created by following naming compounds for corresponding process model IDs, so that the complete name can be written as: `FR_ + {fragmentId} + {parentFeatureName}`. One of the advantage of separating commonality and variability specification is that there may be more than one variability model for each base model. Hence, process modelers may associate as many variability models as they require to an unique base model even representing multiple perspectives for different stakeholders. Further, pre-established fragments can be reused within different variability models when required. In such variability models, all the variability need to be managed considering not only non-context features and process related variability, but also data variety and physical environment (context variability) in which each DSPL operates.

4.2.2 *Context Variability*

As context information is considered the cornerstone for context-aware and self-adaptive systems, managing the context data in the wind farm DSPLs turns into a primary concern. From the variability perspective, traditional feature models do not distinguish between context and non-context features, but the emerging popularity of dynamic variability techniques has leveraged the role of context features. In this light, some research work introduced the notion of context variability [17, 3] as a relevant technique to model contextual awareness of those systems that model their context properties using a software product line approach [27]. Consequently, context features are highlighted in features models and classified as a separated but related branch in the feature tree. The identification and representation of the context properties of a system using software variability pertains to context analysis or context feature modeling techniques.

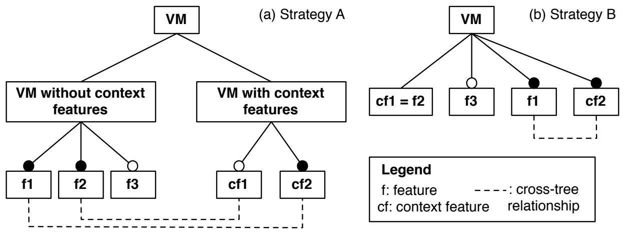


Figure 5: Context variability modeling strategies.

Based on our previous work [9], we describe the following two strategies that can be utilized to represent the context variability (subsumed in the variability model of Figure 4) of systems that follow a product line approach.

Strategy A: Two variability models. As depicted by Figure 5 (a), in this strategy two feature models (variability models) are separated - the one for the system properties of the product family (i.e. non-context and process variability related features from Figure 4), and the other as a branch of the main feature model, which classifies and represents the context properties only (i.e. context variability elements).

Both disjointed variability models could contain diverse constraints, ranging from context feature/non-context feature restrictions to non-context feature/context feature cross-tree relationships. This allows for separation of concerns by separating context and non-context features and reusability so that feature models can be easily replaced by others if needed. However, the main disadvantage, especially in the case of large feature models, is the high number of dependencies between context and non-context properties that could appear in both feature trees, reducing visibility and effectiveness of variability conditions.

Strategy B: A single variability model. As depicted by Figure 5 (b), this strategy uses only one feature model to represent context and non-context properties.

Here, even if the reusability of context properties is lower than in `Strategy A`, the number of dependencies between context and non-context features is reduced. In this approach features can be classified under different types to organize related features, e.g., using super-types description

[9]. However, this strategy requires a bit more modeling effort from the designer to represent all the features under a single feature model, as implicit dependencies between features may exist. The left part of Figure 5 shows two related feature models with context and non-context features in two separate branches (i.e. `Strategy A`) and the dependencies between them. In the right part of Figure 5 we merge both feature trees in one single model (i.e. `Strategy B`), but only feature `f2` has been made equal to context feature `cf1`, while the dependency between feature `f1` and context feature `cf2` persists.

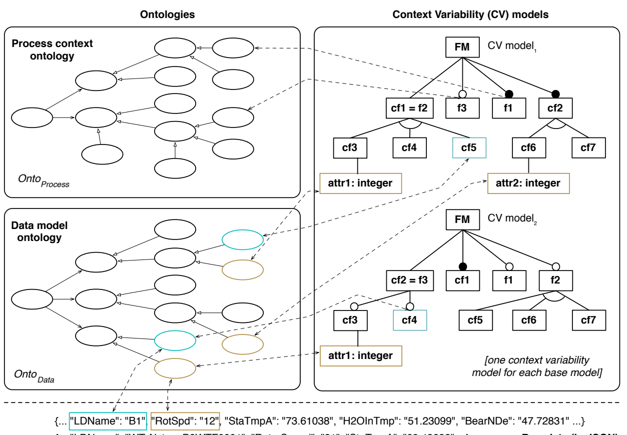
In the following, we have employed the `Strategy B` to model the context variability of wind farms, since we perceive that it is easier to map only one feature model with the ontology (see Figure 6), as well as to manage the process variability of such context properties.

4.2.3 *Context Domain Models*

Different context data may belong to the same context feature, i.e., context data models may differ syntactically and semantically from each other. For instance, while Eco100 turbines collect rotor speed value in `RotorSpeed` variable, `RotSpd` is utilized by Eco80 turbine models for the same purpose. The terminology might be different and manifold context variable names may belong to the same context feature (see Figure 6). Similarly, data ambiguity can arise from the same variable names that stakeholders use for different purposes (e.g. in Eco100 the `LDName` variable sends the full name of the logical device as context data, while in Eco80 `LDName` contains the short name). Such heterogeneity may cause unexpected behavior in data processing and mapping, allowing identical data values to be sent within a different variable name. In context-aware process variability the management of data ambiguity is crucial since process variant customization is context sensitive, i.e., the resolution could differ from context to context depending on domain context data values and feature model constraints. Therefore, we need a consistent way of mapping which context data model variable/value pairs operate at each context feature in a feature model and remove data ambiguity, incompatibility and inconsistency by defining domain specific terminologies to describe the syntax and semantics.

Such heterogeneity of the different underlying analytics processes and context data variety and dynamicity needs to be abstracted to model the domain knowledge and define the semantics and relationships among them. This abstract model is defined using ontologies. This type of representation has been widely accepted as a conducive method for domain modeling (knowledge vocabulary) and reasoning, with low impact on scalability and performance (e.g. in IoT knowledge representation [28]), enabling views on ontologies [11], which means that *"this mapping can give semantics to potentially overlapping feature models, whereas feature models allows scoping and configuring ontologies from different perspectives"*. We distinguish 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 in our approach (e.g. in the NoSQL data-store), including different entity types and their relationships. For instance, each wind farm component type from Figure 2 may have an unique data model (e.g. represented by a JSON Schema[5]) and integrated in the data model ontology, where an intentional description of the onshore/offshore data classes and instances are characterized. Such context information is exploited to correlate the expected contextual data attributes in a context variability model to those currently being experienced, as illustrated in Figure 6. For instance, `RotSpd` data is mapped to ($Onto_{Data}$) data property which also links the `attr1` context feature attribute. Feature attributes are used to map dynamic 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.



```
{... "LDName": "B1", "RotSpd": "12", "StaTmpA": "73.61038", "H2OInTmp": "51.23099", "BearNDe": "47.72831" ...}
{... "LDName": "WT Alstom B2WTF0001", "RotorSpeed": "21", "StaTmpA": "68.43233" ...}
```

Figure 6: Context variability modeling approach (using `Strategy B`).

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[6] standard which permits to correlate different data variable/value pairs to an equivalent ontology concept, as shown in Figure 7. 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. Eco100, Eco80) by mapping them to 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.

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

```
{
  "@context":
  {
      "LDName": {
          "@id": "http://alstom.com/ontoData/windTurbine",
          "@type": "@id"
      },
      "RotSpd": "http://alstom.com/ontoData/wt/rotorSpeed",
      "StaTmpA": "http://alstom.com/ontoData/wt/generatorTmp",
      "H2OInTmp": "http://alstom.com/ontoData/wt/waterTmpInlet",
      "BearNDe": "http://alstom.com/ontoData/wt/nonDriveAndBearingTmp",
  },
  "LDName": "B1",
  "RotSpd": "12",
  "StaTmpA": "73.61038",
  "H2OInTmp": "51.23099",
  "BearNDe": "47.72831"
  }
```

Figure 7: Excerpt of a JSON-LD inline context data.

Process context ontology ($Onto_{Process}$). In conjuction 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 WFaaS 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 feature model and also a *hasType* for representing the type of feature model (feature combination strategies from Figure 5).

Based on those mappings, the process selection service is capable of retrieving base models for a given WFaaS request.

5. DISCUSSION

The presented approach is currently being developed on top of the LateVa framework [23] to manage context data variability for context-aware process variability at runtime. Therefore, we still do not have enough evidence of about the effort required during the modeling or even the scalability/performance issued from its adoption. Nevertheless, we cope with context data ambiguity for service-based DSPLs by modeling process variability, context variability and domain modeling, in order to manage raw data variety and heterogeneity coming from different wind farms (ICSs and SCADA systems).

As part of our future work, we expect to quantify mentioned issues and validate the presented approach in a realistic industrial wind farm case study, while carefully collecting performance and scalability metrics. Last but not least, we will also expand the presented context variability approach with the aforementioned requirements, such as **R2** for dynamic behavior of features and **R3** multiple and dynamic binding support.

6. RELATED WORK

The notion and challenges of DSPLs described in [16, 18] have led to a number of techniques that can be used in isolation or combined to achieve the necessary dynamicity of runtime changes and multiple bindings.

One of the challenges that makes the difference between conventional SPLs and DSPLs is the ability to manage the structural variability at runtime. As many autonomous and self-adaptive systems demand changes in the variability at runtime, dynamic variability becomes one of the central mechanisms for DSPLs to manage the adaptivity of systems more efficiently [6]. Systems that demand adding a feature at runtime or the simple activation and deactivation of systems features require of specific runtime variability techniques [7]. Furthermore, pervasive, cyber-physical and smart systems that exploit context properties require representation mechanisms to model the context variability [17, 9]. Consequently, modeling the context properties using context variability models becomes a challenge for DSPLs that need to specify which context properties must change and in which order, according to different context scenarios and reconfiguration alternatives. Some experiences combining context-aware information with variability models using a DSPL approach can be found in [1, 25]. Other authors combine context information gathered by sensors in smart home systems [10] model such context properties using software variability and runtime variability mechanisms typical from DSPL approaches to reconfigure the system when the context changes and providing rollback capabilities when needed. However, although all of them provide useful mechanisms to partially support some of the posed requirements for DSPLs, none of them describe a solution for managing and modeling context variability in service-based DSPLs.

Closely related to this paper, some authors highlight the role of DSPLs for service-based systems, where reconfigurations and rebinding activities are used for the variability of business services with context properties and dynamic orchestration models [19, 24, 21, 5]. The problem for adaptive and flexible service compositions has been studied by some works [4], where binding and re-binding of composite services, often driven by Quality of Service (QoS) goals [8], is an important concern for IT services to determine the best and optimized service sequence. Therefore, capturing and modeling the variability of business process models and their orchestrations [20, 15] for supporting dynamic business process adaptation [29] results key for DSPL approaches that use context information and runtime variability mechanisms to determine the most suitable service selection and composition during service reconfiguration operations [2]. Other experiences define the notion of *smart workflow* to compose different user tasks depending on the context information or data gathered from different inputs (e.g.: user, sensors) and they apply their approach to mobile workflows [13]. Compared to those latter approaches, in this paper we have presented a methodolody for context variability and design building blocks of an approach for runtime configuration of processes in context-aware systems.

7. CONCLUSION

The ambiguity and heterogeneity of incoming data of wind farms need of both a significant context-aware modeling effort and runtime variability techniques to cope with dynamic changes. In this paper, we first outlined the main requirements for supporting context variability for service-based DSPLs, addressed the modeling problems of wind farms context variability and suggested dynamic variability as a way to anticipate to the evolution of changes when data or new context features demand a modification of the structural variability. Initial experiments have been done on top of our LateVa toolkit which supports runtime variability and multiple binding requirements of process-based services. However, we believe the natural integration of process-based variability with context features (context variability) can simplify the management of adding/removing or changing the wind farms context properties.

Acknowledgements

This work was partially funded by the Spanish Ministry of Economy and Competitiveness (Ref. TEC2011-28397).

8. REFERENCES

[1] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J.-P. Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *MODELS Workshops*, 2009.

[2] G. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz. Dynamic adaptation of service compositions with variability models. *JSS*, 2013.

[3] R. Ali, Y. Yu, R. Chitchyan, A. Nhlabatsi, and P. Giorgini. Towards a unified framework for contextual variability in requirements. In *IWSPM*, pages 31–34, Sept 2009.

[4] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE TSE*, 33(6):369–384, June 2007.

[5] L. Baresi, S. Guinea, and L. Pasquale. Service-oriented dynamic software product lines. *Computer*, 45(10):42–48, Oct 2012.

[6] N. Bencomo, S. Hallsteinsen, and E. Santana de Almeida. A view of the dynamic software product line landscape. *Computer*, 45(10):36–41, Oct 2012.

[7] J. Bosch and R. Capilla. Dynamic variability in software-intensive embedded system families. *Computer*, 45(10):28–35, 2012.

[8] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. A framework for qos-aware binding and re-binding of composite web services. *J. Syst. Softw.*, 81(10), 2008.

[9] R. Capilla, Ó. Ortiz, and M. Hinchey. Context variability for context-aware systems. *IEEE Computer*, 47(2):85–87, 2014.

[10] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Prototyping dynamic software product lines to evaluate run-time reconfigurations. *Science of Computer Programming*, 78(12):2399 – 2413, 2013.

[11] K. Czarnecki, C. H. Peter Kim, and K. T. Kalleberg. Feature models are views on ontologies. In *SPLC*, pages 41–51, 2006.

[12] M. Döhring, H. A. Reijers, and S. Smirnov. Configuration vs. adaptation for business process variant maintenance: An empirical study. *Information Systems*, 39:108–133, 2014.

[13] P. Giner, C. Cetina, J. Fons, and V. Pelechano. Developing mobile business processes for the internet of things. *Pervasive Computing, IEEE*, 9(2):18–26, April 2010.

[14] I. Groher and M. Voelter. Expressing Feature-Based Variability in Structural Models. In *Workshop on Managing Variability for Software Product Lines*, 2007.

[15] A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: The provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(6-7):519–546, November 2010.

[16] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, April 2008.

[17] H. Hartmann and T. Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *SPLC*, pages 12–21, 2008.

[18] M. Hinchey, S. Park, and K. Schmid. Building dynamic software product lines. *Computer*, 45(10):22–26, 2012.

[19] P. Istoan, G. Nain, G. Perrouin, and J.-M. Jezequel. Dynamic software product lines for service-based systems. In *CIT*, pages 193–198, 2009.

[20] M. Koning, C.-a. Sun, M. Sinnema, and P. Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Inf. Softw. Technol.*, 51(2):258–269, Feb. 2009.

[21] J. Lee, G. Kotonya, and D. Robinson. Engineering service-based dynamic software product lines. *Computer*, 45(10):49–55, 2012.

[22] S. Meyer, K. Sperner, C. Magerkurth, and J. Pasquier. Towards modeling real-world aware business processes. In *WoT*, pages 8:1–8:6, 2011.

[23] A. Murguzur, X. de Carlos, S. Trujillo, and G. Sagardui. Context-aware staged configuration of process variants@runtime. In *CAiSE*, 2014.

[24] C. Parra, X. Blanc, and L. Duchien. Context awareness for dynamic service-oriented product lines. In *SPLC*, pages 131–140, 2009.

[25] K. Saller, M. Lochau, and I. Reimund. Context-aware dspls: Model-based runtime adaptation for resource-constrained systems. In *SPLC Workshops*, pages 106–113, 2013.

[26] M. Schlechtingen, I. Santos, and S. Achiche. Using data-mining approaches for wind turbine power curve monitoring: A comparative study. *Sustainable Energy, IEEE Transactions on*, 4(3):671–679, July 2013.

[27] E. Vassev and M. Hinchey. Awareness in software-intensive systems. *Computer*, 45(12):84–87, Dec 2012.

[28] W. Wang, S. De, R. Tönjes, E. S. Reetz, and K. Moessner. A comprehensive ontology for knowledge representation in the internet of things. In *TrustCom*, pages 1793–1798, 2012.

[29] Z. Xiao, D. Cao, C. You, and H. Mei. Towards a constraint-based framework for dynamic business process adaptation. In *SCC*, pages 685–692, 2011.