From Adaptive Business Processes to Orchestrated Microflows

Andreas Daniel Sinnhofer[1,2], Roy Oberhauser[3], and Christian Steger[1]

[1] Institute of Technical Informatics, Graz University of Technology, Graz, Austria,
a.sinnhofer@alumni.tugraz.at, steger@tugraz.at,
[2] NXP Semiconductors, Gratkorn, Austria,
[3] Computer Science Department, Aalen University, Aalen, Germany.
roy.oberhauser@hs-aalen.de

Abstract. Nowadays, businesses with focus on consumer-products are challenged by short production cycles, high pricing pressure, and the need to deliver new features and services in a regular interval. Currently, businesses are tackling these challenges by automating their business processes, while yet trying to be flexible by introducing methods for process variability modeling. However, for larger processes and variability models, it becomes difficult to consider, maintain, and optimize all process variations in the various execution contexts. In software development, highly agile requirements are usually tackled with a flexible microservice architecture. Nonetheless, the fast-changing service landscape is often not fully reflected in the underlying business processes, leading to inefficiency and loss of profit. With this work, we extend our framework for process variability modeling with concepts of Microflows, allowing agile business process modeling and orchestration while utilizing the full flexibility of underlying microservices. In addition, we present a case study, showing how this approach is used in the context of an IoT application.

Keywords: Business Processes, Workflow Management Systems, Microservices, Software Product Lines.

1 Introduction

Today's society is heavily driven by an ever-changing and interconnected world. Heavily shaped through the digital transformation, new technologies like Internet of Things (IoT) are pushed as a solution for our daily problems. Generally speaking, IoT refers to the connection of our everyday objects with a network like the Internet [1]. Each of these devices is equipped with different kinds of sensors to observe its environment, making the device a smart object. In combination with embedded systems, IoT promises to increase the quality of our daily lives by taking over simple tasks like controlling the room temperature or cooking coffee. For businesses, this means that feature-rich systems are demanded by the customers, with the ability to unlock new services and features on a regular basis. Consequently, new methods are investigated on how to efficiently model process and product variability [2].

Business Process (BP)-oriented organizations are known to perform better regarding highly flexible demands of the market [2]. A workflow describes the automation of a BP by applying a set of procedural rules [3]. By using a workflow management system (WfMS), workflows are defined, created, and managed. However, while adaptive WfMS can handle a certain degree of flexibility, they usually require manual intervention and rework. As such, context-aware BP modeling techniques were introduced to cope with fast changing requirements [4]: by analyzing the context states of the environment and by mapping the suitable BPs to their related software systems, flexibility is gained that can then be used to automatically select, adapt and execute a process variant. Problems of this approach are that such systems are often developed independently from each other [2]. Consequently, operating and maintaining these variants can lead to unnecessary overhead.

Since recent years, software services or microservices are often utilized in software development to support a digital automation [5]. As described by Fowler and Lewis [6], microservices provide an agile and loosely-coupled partitioning of business capabilities into service implementations. Each service is individually evolved, deployed, and executed. However, due to the rising number of services and automation expectations, approaches for a dynamic webservice orchestration are needed. Service orchestration can be split into: (1) centralized approaches, e.g., using flow descriptions – and (2) decentralized approaches, e.g., using collaborative interaction of services [7]. However, an integration of these approaches to dynamically plan and invoke processes remains a challenge.

The reuse of software components is an important step for an industrial company to survive in a flexible and competitive market [8]. By applying a microservice architecture, the focus on reuse is often lost: teams focus on delivering work quickly and independently, often avoiding dependencies to other teams or to shared code that is not maintained by them. However, we think that software reuse is essential also in the context of webservice development to raise software quality and to minimize time-to-market. Software Product Lines (SPLs) have proven to be highly effective in reusing software artifacts [9]. The most critical phase during the design of an SPL is the identification of the variable parts and the common parts of a product family [8]. In the context of webservice architectures, SPL can be beneficial for the implementation of common libraries that are shared accross different teams, and to define common solution architectures. However, an integrated view on BPs is often missing, leading to inefficient development overhead [10].

With this work, we focus on extending our framework [11,8,2] with capabilities of Microflows [12] to allow an automatic orchestration of microservices based on annotated processes. In a nutshell, the annotated process model describes the pre-conditions and constraints that must be met to execute the process and the post-conditions that are a consequence of executing the process. Pre-conditions can be as simple as a specific input parameter, or constraints about the execution sequence (e.g., before execution of process X or after Y). By applying our framework for combined variability management, we enforce a strong link

between the developed software artifacts and the business processes in which they participate, take full advantage of SPL Engineering techniques for reusing software artifacts, and enable high variability within workflows.

This work is structured in the following way: we present related work in Section 2. Section 3 summarizes the basic concepts of SPL Engineering (SPLE), Business Process Modeling (BPM), as well as Microflows that we have applied in this paper. Section 4 summarizes our approach for combined variability modeling of business processes and software architectures, and how automated process orchestration can be applied. In Section 5, we describe how the proposed framework was applied in the context of an IoT case-study. And finally, Section 6 concludes this work.

2 Related Work

IBM defines a Microflow as short-lived BPEL processes [13]. However, in this work, we use the definition found in [12], defining it independent of any specific Business Process Management System (BPMS).

Web-Service composition [14], provides a survey of prototypes and standards for composition of webservices. Rajasekar et al. [15] presents a technique to orchestrate microservices based on a distributed event-condition-action rule engine. Rao and Su [16] present a framework for webservice orchestration by using an explicit composite service.

Traditionally, BP modeling languages do not explicitly support the representation of families of process variants [17]. As a consequence, a lot of work can be found which tries to extend traditional process modeling languages with notations to build adaptable process models. Having such a variability modeling for BP models builds the foundation of this work. Thus, related work which utilizes similar modeling concepts is presented in the following:

Derguech [18] presents a framework for the systematic reuse of process models. In contrast to this work, it captures the variability of the process model at the business goal level and describes how to integrate new goals/sub-goals into the existing data structure. The variability of the process is not addressed in this work.

Gimenes et al. [19] presents a feature-based approach to support e-contract negotiation based on webservices (WS). A meta-model for WS-contract representation is given and a way is shown how to integrate the variability of these contracts into the BPs to enable process automation. It does not address the variability of the BP itself but enables the ability to reuse BPs for different e-contract negotiations.

While our framework to model process variability reduces the overall process complexity by splitting up the process into layers with increasing detail, the PROVOP project [20,21,22] focuses on the concept that variants are derived from a basic process definition through well-defined change operations (deletion, addition, moving of model elements, or the adaptation of an element attribute). In fact, the basic process expresses all possible variants at once.

The work of Gottschalk et al. [23] presents an approach for the automated configuration of workflow models within a workflow modeling language. The term workflow model is used for the specification of a BP, which enables the execution in an enterprise and WfMS. The approach focuses on the activation or deactivation of actions and thus is comparable to the PROVOP project for the workflow model domain.

La Rosa et al. [24] extends the configurable process modeling notation developed from [23] with notions of roles and objects, providing a way to address not only the variability of the control-flow of a workflow model but also of the related resources and responsibilities.

The Common Variability Language (CVL) [25] is a language for specifying and resolving variability independent from the domain of the application. It facilitates the specification and resolution of variability over any instance of any language defined using a MOF-based meta-model. CVL-based variability modeling in combination with a BP model with an appropriate model transformation could lead to similar results as presented in this paper.

The work of Zhao and Zou [26] shows a framework for the generation of software modules based on BPs. They use clustering algorithms to analyze dependencies among data and tasks captured in BPs.

3 Background

This section summarizes the basic concepts of SPLE, BP Modeling, and Microflows that are applied in this work. This section is based on our previous publications that form the foundation of this work [11,27,2,12,5].

3.1 Software Product Line Engineering (SPLE)

SPLE applies the concept of product lines to software products. Thus, SPLE delivers diverse, high-quality software products of a product family in a short time and at low price [9]. Instead of writing software for every individual system, SPLE makes use of software components (domain artifacts) which are diversified and combined in order to generate the final software product. As described in [9,28], the SPLE can be split into two main phases, the *Domain Engineering* and the *Application Engineering*:

During the domain engineering, the domain is modeled and the variabilities and the commonalities of the according domain are identified and reflected in the implemented domain artifacts. Domain artifacts are reusable development artifacts like the software architecture, or software components and their corresponding unit-tests [9].

In the application engineering phase, the final software products are created by combining and diversifying the domain artifacts which were implemented in domain engineering. The main goal of application engineering is to maximize reuse of domain artifacts. Additionally, implemented logic usually just consists

of glue-logic between the different domain artifacts, which is often fully generated. Consequently, the rapid creation of high-quality products can be achieved. The degree of domain artifact reuse depends to a large extent on the application requirements. Hence, a major concern of the application engineering is the detection of deltas between the application requirements and the available capabilities of the SPL. During the lifetime of a product line, these deltas often result in additional domain artifacts which can be reused in future products.

In the context of webservices, SPLE provides capabilities for reusing common implementations (like user-interfaces, library implementations, and others), leading to a more robust and mature software base for each service.

3.2 Business Process Modeling

Business Processes (BPs) are a specific sequence of activities or (sub-) processes which are executed in a dedicated sequence to produce output with value to the customer [8,29]. In this work, we use the modeling paradigm defined by Oesterle [30]: The BPs are split-up into different layers until the microscopic level is reached. This is achieved when all tasks are detailed enough so that they can be used as work instructions. The top level (macroscopic level) is a highly abstract description of the overall process, while each subprocess is further described in lower levels. Consequently, higher levels of the process are usually independent of the production facility, the infrastructure, and environmental specifics. Thus, the higher level is more stable with respect to changes and can be reused in different contexts and production environments. The microscopic levels, however, require adaptation to be reused in different contexts.

Variability of such process structures can be modeled through a variable process structure (i.e. by adding/removing activities in a process) or by replacing sub-processes with different ones. In the scope of this work, we use BPMN (Business Process Model and Notation) [31] as a modeling notation for BPs, but the general framework is not limited to BPMN as long as the modeling notation supports concepts like events, activities, responsibilities, data objects (used to describe inputs and outputs), and control flow elements (to model, e.g., choices and parallel executions).

3.3 Microflows

A Microflow can be seen as a short-lived process execution, which is defined by an execution of webservices. The following enumeration lists the important principles used for modeling and orchestrating Microflows in the context of this work (for a full list of principals, see [5]):

Microservice Semantic Self-description Principle: A microservice provides sufficient metadata to support autonomous client invocation.

Client Agent Principle: We chose Belief-Desire-Intention (BDI) agents for the client realization, were belief is provided via knowledge, desire via goals, and intention are represented in the resulting workflow.

Graph of Microservices Principle: microservices / workflow activities are represented as nodes in a graph. Each node is annotated with properties. Edges depict the directed connections between the nodes.

Microflow as Graph Path Principle: A directed graph of nodes corresponds to a workflow, and is determined by an algorithm applied to the graph. During the workflow execution, each node and respectively the underlying microservice is executed, with inputs and outputs as specified in the annotated microprocess.

Declarative Principle: Any workflow requirement specification take the form of goal and constraint modeling statements. It contains the starting microservice, the end microservice, and additional constraints that must be met during the workflow execution.

Path Weighting Principle: Any edge of the microservice graph can be weighted with a potentially dynamic cost which helps in quantifying and comparing path alternatives. As such, the navigation from one node to another node can be dynamically adjusted based on collected process execution data (like response time).

4 From Adaptive Processes to Orchestrated Microflows

In this section, we give details on how we model variability of BPs, how to generate BP variants, and how to orchestrate BPs using BDI agents.

4.1 Managing Variability in Business Processes

A detailed description of our framework for modeling variability of BPs and software architectures can be found in [2]. In the scope of this work, we will briefly summarize the concept and present the extensions to the original framework. The extensions allow a combined variability modeling not only for higher-order processes, but also for processes which make use of webservices to trigger actual actions on the systems. Figure 1 shows the overall extended framework. The original framework consists of two essential parts: The *Process Variability Framework* and the *Product Software Product Line*, while the extended framework introduced the *Microflow Orchestrator*.

Process Variability Framework (c.f. [11,27,32]) is a framework used to manage variability of BPs by applying concepts of SPLE. As such, maintaining and evolving process variants of a family is done by automatically applying changes to the model. These changes are automatically propagated to all process variants. By using rich constraint checking engines and code generation methods, it is ensured that the generated process variants are consistent. The starting point for the variability modeling is the process of domain modeling and process-template creation: during this process, the requirements of the domain are analyzed and appropriate process templates are created. In addition, variation points (VPs) are identified and reflected in the variability model. The framework supports variability management of a whole process hierarchy, starting at the top level

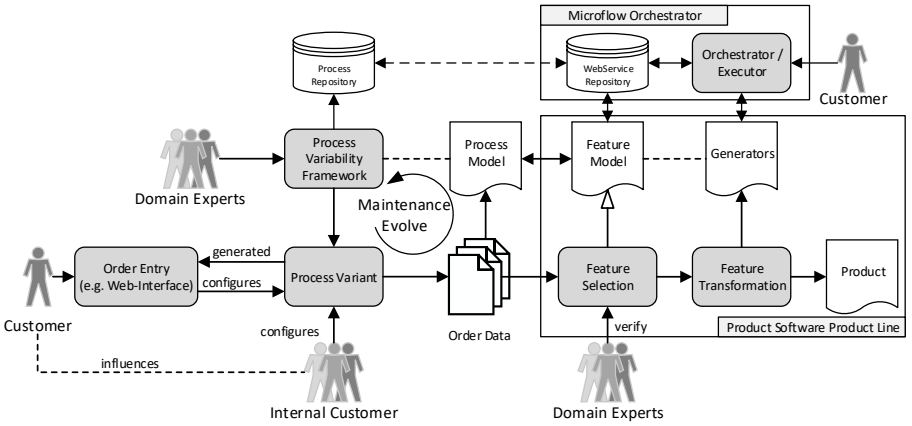

Fig. 1. Overall framework for combining process variability and product variability based on [32], extended with runtime service aspects using Microflows.

(macroscopic processes) to the lowest level (microprocesses). On each level, certain transformations can be applied which are: replacing non-atomic activities / subprocesses with different ones, adding and removing activities and resequencing process steps.

Product Software Product Line is a SPL which allows configuration and generation of software products during product order: the variable parts of the process variants are mapped to the variability model of the product line using identifiers, ensuring a traceable link between both models. To support the process of modeling, we have defined mapping rules whereby standard BPMN processes can be translated into a basic feature model skeleton of the product: activities are reflected by features, and process inputs are reflected as configuration options [2]. By analyzing the structure of the process model, the system can also identify which features are mandatory, which ones are optional, and what basic constraints are defined between them. In this manner, we have shown in [8] that product configurations can be automatically generated based on, e.g., an order defined by the customer.

To summarize, the original framework provides a configurable solution, which allows the configuration of adaptive BPs, for instance during product order, based on customer/stakeholder inputs. In addition, a SPL is automatically configured using the input configuration of the (order) process, and the final software products are automatically generated. One limitation of this system is that some of the actions still required manual interaction, like triggering the source-code generator, uploading files to a server, etc.

With this work, we want to extend the scope of the framework, in order to support a WS-oriented business to automatically create and execute Microflows without the need for manual interaction during the execution. To allow this, we assume the following:

- An atomic activity can be modeled as a WS (microservice) call
- A *microprocess* conists of a sequence of atomic activities.
- A *microprocess* defines meta-information on how and when it can be triggered (see Section 4.2 for details).
- A Microflow can be modeled as a sequence of *microprocess* executions.

The concept of *microprocesses* consisting of multiple microservice executions is purely used for grouping and modeling purposes (i.e., allowing a single model to generate multiple WS variants) and could be seen as a smaller Microflow. If the provided desire and conditions are met, a Microflow can be created dynamically on the fly by defining a start condition, an end condition, and additional constraints that must be met. In order to support a flexible design of *microprocesses*, we apply our Process Variability Framework to model process variants and to publish them in a repository. By applying the concept of combined variability modeling, we can generate microservice descriptions which must be implemented by the developers, while still providing a concept for reusing shared functionality.

4.2 Orchestration of Microprocesses based on Annotated Process Models

By using the concept described in Section 4.1, a repository of business microprocesses is generated and available. To support their automated orchestration, each microprocess is annotated with additional meta-information, allowing an algorithm to calculate possible paths through the set of available microprocesses in order to achieve a defined goal. In this work, we specifically make use of a shortest path algorithm, but the overall framework is not limited to this algorithm. The set of annotations required to calculate possible paths can be summarized as:

- *Path weight*: The path weight gives an indication on how "expensive" it is to execute a specific microprocess. By allowing dynamic path weights (e.g., updated due to process execution logs, or different weights based on different execution environments), it is possible to gain flexibility and increase the quality of service.
- *Constraints*: A list of constraints that must be met. Our framework makes use of the following classes of constraints:
  - *Input Parameter*: Input parameters that are necessary in order to execute a process. These parameters can either be provided by a client, or can be the outcome of another microprocess execution.
  - *Output*: The outcome(s) of a process execution. To allow flexible modeling of process executions, it is also possible to define meta-data output that is only used for planning purposes.
  - *Before Node*: Constraint used to model that specific microprocesses must be executed before the execution of this microprocess. This can also be modeled using specific types of input parameter and output values, but by using the concept of a *Before Node*, more flexibility is gained.

- *After Node*: Constraint used to model that specific microprocesses must be executed after the execution of this microprocess. Identical to a before node constraint if defined in reversed order.
- *Exclude Node*: Constraint to indicate that a specific microprocess must not be executed at all (neither before, nor after).

In our approach, we do not fully dynamically re-calculate the next node during process execution. For a simpler and more lightweight framework: we make use of a planning phase where a BDI agent takes a goal and additional constraints to find an appropriate schedule of microprocess executions. Mandatory constraints for the agent are the selection of the start node, the end node, and may optionally contain additional microprocesses that must be invoked during the execution. Only in case where the execution of a microprocess fails, we recalculate an alternative path by invoking the planning phase again. By doing this, we can ensure that at least one valid alternative Microflow path exists which fulfills the requirements, prior to invoking an error recovery.

In a nutshell, the currently implemented path finding algorithm performs the following steps: First, it identifies the start node and determines which nodes can be directly reached while meeting the defined constraints. Secondly, for each direct connection, the algorithm basically calls itself recursively with the new node being the start node, and it updates its constraints. The updated constraints basically contain the output(s) from the former start node as additional available input parameter, and a call history containing the start node. This is necessary to prevent visiting the same node twice. The algorithm continues by recursively calling itself until a list of possible execution paths are collected. These paths are then evaluated using the defined path weights and the shortest one is taken. In case multiple paths converge to the same weight, implementations can implement various strategies like round-robin, or choosing a random one.

5 Case Study

For illustration purposes, we look at an exemplary IoT use-case which we are currently working on with our research partners: In a home automation scenario, various IoT devices and actuators are installed. The overall concept is illustrated in Figure 2. Some IoT devices may be installed in a fixed location, with a power supply, while others may be battery powered and mobile (like a smartphone). Additionally, devices may be produced by different vendors. From a technical perspective, each IoT device implements some functionality like reading sensor values, returning the health status (battery level), and enabling/disabling output ports. Each device is connected either directly, or via a gateway to an IP network having access to a so-called *Service Platform*. For this example, we assume that this *Service Platform* is a local reachable Server in the LAN network. Specific services can be installed to the *Service Platform* that can be utilized by the customer. Each service accesses functionality of the IoT devices and/or the

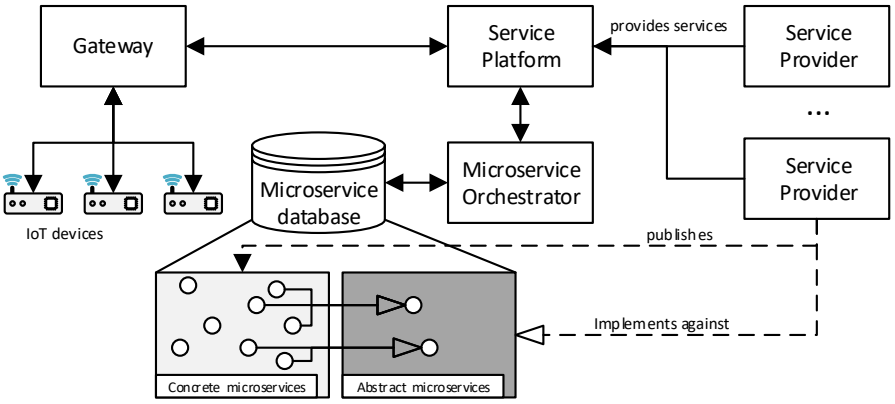

Fig. 2. System overview of the case study.

actuators. For development purposes, the *Service Platform* provider (i.e., a company offering a platform for IoT Services) makes use of our introduced framework for modeling and implementing small microservices for accessing functionality of the IoT network. Consequently, a *Service Provider* does not need to implement code to access functionality of the linked devices. However, a provided service uses abstract webservice invocations to get and set information in the IoT network. Consequently, the *Service Platform* is responsible for WS orchestration and execution based on the desire and constraints defined by the service.

The *Service Platform* may provide abstract microservices like readSensorValue and setOutputValue, which internally are represented by a sequence of microservice calls like connect, authenticate, readSensorValue, and setOutputValue. Each of these microservices can be implemented in several different ways depending on the used hardware and supported protocols. For example, if a *Service Provider* implements a service for regulating the room temperature (i.e., switching a heating element on or off, depending on the current room temperature), the service formulates the desire to set the output voltage of an outlet (setOutputValue), with the constraint of first sensing the room temperature and afterwards doing some calculation (a micro-service provided by the service provider).

5.1 Example Microflow: Measuring the room temperature

In the following, we will take a closer look at an example. First, we start with a simple case: The network consists of identical IoT nodes, having the same capabilities (i.e., each node is equipped with a temperature sensor). All of them are permanently powered. From a high level perspective, the *Service Provider* models his business process as depicted in Figure 3: In a regular interval, he requests the current room temperature from the IoT network, decides with a very basic decision if the temperature is above or below a certain threshold ($23°C$), and

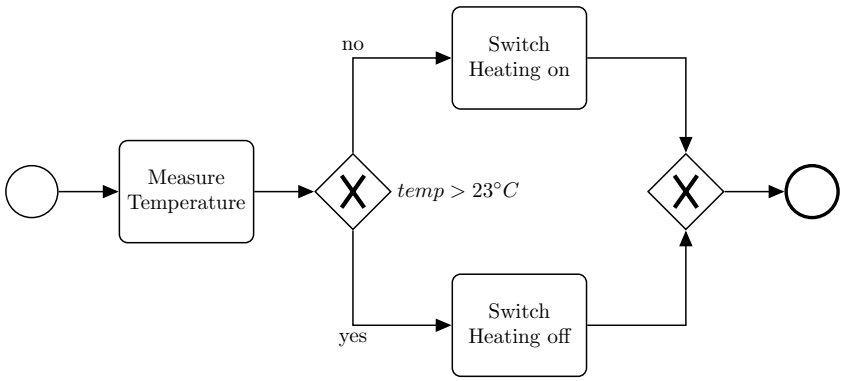

Fig. 3. Business Process for regulating the room temperature (Macroscopic level).

switches the heating system on or off. To keep this example simple, we consider the room temperature regulation process on the Macroscopic level as a "normal" BP, while each sub-process of the BP is modeled as a Microflow. In the following, we will take a look at the details of the *Measure Temperature* microflow. The microservice database consists of the following microservices which are of interest:

- **Connect**: Is an abstract microservice which takes care of connecting to an IoT device using a protocol that is supported by the device. The actual implementation of this microservice takes care of handling device and protocol specific aspects.
- **Authenticate**: Is an abstract microservice which takes care of authenticating the *Service Provider* to the device. The actual implementation of this microservice depends on the communication protocol and device capabilities. In the following we group authentication protocols based on symmetric and asymmetric cryptography.
- **Read Temperature**: Is an abstract microservice which takes care of reading the current temperature from a device. The actual implementation of this microservice depends on the communication protocol and the authentication method (e.g., command sent via a secure communication channel).
- **Close Connection**: Is an abstract microservice which takes care of closing the communication to an IoT device. The actual implementation of this microservice depends on the communication protocol and the authentication method.

The particular implementations of the above mentioned microservices are provided by the IoT device/system manufacturer. This allows *Service Providers* to define services independent of the used hardware. For this example, we consider that a connection to the IoT devices can be established via Bluetooth, WiFi, or via Zigbee. For each of these communication protocols, we consider two possible authentication methods/protocols: one based on asymmetric cryptography

Table 1. Overview of the constraints for each microservice

	Input Parameter	Output	Before Node	After Node
Connect*	deviceAddress	connection	-	-
Authenticate*	connection	channel	-	-
Read Temperature	channel	temperature	Close Connection	-
Close Connection	-	-	-	-

* Each implementation defines its own input and output types to allow an automatic orchestration. For illustration, only the abstract input / output types are shown in this table.

(like Transport Layer Security (TLS)), and one based on symmetric cryptography (like Secure Channel Protocol (SCP)). We will not look into the details of the *Read Temperature* or *Close Connection* webservices. The constraints of the individual microservices are illustrated in Table 1. Note that each concrete implementation of the abstract microservice can define different input and output parameter. For example, a webservice connecting to the device via WiFi may require an IP address and a port number as input, while a connection via Bluetooth may only need the Bluetooth device address as input. In addition, by using different output types, the orchestrator can take care of selecting the correct subsequent microservice in case there are execution dependencies. For example, if a specific authentication method requires a dedicated connection interface, the connection interface could produce an output that can only be consumed by this special authentication method. Other possibilities would be to define a more fine grained constraint model in which each concrete service implementation defines specific *Before Node* and *After Node* constraints.

The desire of this Microflow is measuring the *temperature* which is the output data of the *Read Temperature* microservice. Thus, the path finding algorithm is able to calculate a Microflow fulfilling the desire: *Read Temperature* requires a channel as input parameter. Consequently, *Authenticate ** must be executed prior to reading the temperature, which requires *Connect ** to be executed before to establish a connection. And finally, after reading the temperature value – and returning it to the *Service Provider* – the *Close Connection* webservice has to be called. Noteworthy is the input parameter of the *Connect ** webservice: The deviceAddress must be provided by the *Service Platform* which keeps track of all the registered devices in the network, as well as their capabilities. The generated Microflow is illustrated in Figure 4. For Microflow execution, the Microflow Orchestrator receives a list of devices, calculates the path for each device to produce the desire (the temperature value), rates the generated variants according to the path weight, and selects the path with the lowest path weight. Various strategies can be used in case multiple paths lead to the same path weight, like round-robin or randomly choosing a device.

In case a failure happened during the execution of the Microflow, error recovery strategies need to be implemented. In case of IoT use-case, devices tend to terminate connection unexpectedly due to bad signal strength, power failures, or

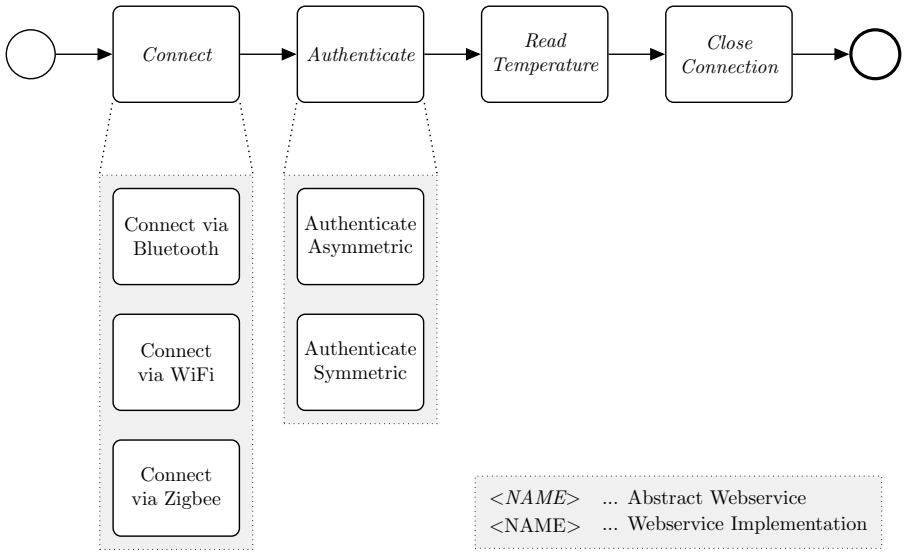

Fig. 4. Schematic overview of the generated Microflow, indicating the pool of concrete service implementations used during execution of the flow.

similar issues. Error recovery strategies can be quite complex, especially if some of the previous execution steps need to be reverted in order to not leave the network in an unstable state. In the context of this example, the strategies can be as simple as a retry mechanism. This means that the current webservice is executed a second time, or the whole Microflow is restarted at the start node. In case a retry also did not result in a successful process execution, the Microflow Orchestrator chooses an alternative route (e.g. measuring the temperature via a different device). If all strategies have failed, the *Service Provider* (i.e., the actual service implementation) is notified via an exception. As such, alternative error-recovery strategies can be implemented on this level as well.

Last but not least, we will shortly discuss the impact of using different IoT devices in a more complex setup and how this impacts the service orchestration: The IoT network consists of various different devices. A number of devices is battery powered, each device can only support one communication protocol, and the equipped temperature sensors are of varying quality. To model these additional constraints, each webservice is annotated with additional meta-data used for path calculation, like: energy consumption, execution time, sensor accuracy, and others. Each of these categories can be individually weighted: If the service requires a highly accurate temperature value, accuracy can be weighted more heavily then others, or if the field of application requires highly energy efficient implementations, energy consumption can be rated more heavily. In the current state of the implemented framework, these path weights are not updated automatically, but only manually by a user, or system administrator. In future

work, we want to investigate possible solutions on how to update these weights automatically by making use of e.g., observed system properties like the received signal strength (to enable or disable communication protocols), the battery state, and others.

To summarize the examples above, the Microservice Orchestrator takes care of identifying the registered IoT devices. During Microflow execution, the Measure Temperature Microflow is expanded with context specific executions. In case a node is not reachable during webservice execution, error-recovery strategies are used which may trigger the path finding algorithm a second time, finding an alternative node to get the room temperature. The presented examples highlight, that the approach helps to create an abstraction for service developers to access functionality of IoT devices without the need of developing vendor specific code. By using complex mechanics for adapting path weights, the system can be optimize according to different strategies, like power consumption or quality of service.

While we heavily discussed microservice orchestration and Microflow execution, we only gave little information on how to create constraint models and how to benefit from variability modeling. In most of the cases, a *Service Provider* is not interested to create such constraint models, nor cope with execution sequences on the microscopic level. The *Service Provider* is usually only interested in reading or setting values of the IoT network. Thus, orchestration of webservices is mostly relevant on a higher abstraction level. For the microscopic process level, the *Service Platform* usually provides convenience models which can also be used as templates for custom services. In this case study, we applied our framework for modeling business process variability [2], but instead of limiting the usage to order processes, we extended it with runtime aspects and custom model transformation engines: The above example was derived from a variability model which is partially illustrated in Figure 5: The feature model shows a generic model of Microflows which are capable of measuring either the temperature or the luminous intensity. Features on higher levels represent abstract webservices (i.e. abstract microprocesses), while child features (i.e., the leafs in the tree) represent the actual implementations.

The *Measure Temperature* Microflow – which was used as example in this case study – was created by selecting the *Temperature* feature and transforming the model. Other selections – like the connection method or the particular authentication method – are not chosen during model transformation, but are selected during Microflow invocation. To allow a transformation of the feature model into the constraint model defined in Table 1, additional information has to be annotated to the feature model: For each feature, we define input parameter and output values. *Before Node*, *After Node* and *Exclude Node* constraints can be partially derived by the dependencies between the specific features: A *requires* dependency indicates a dependency on the sequence of execution. However, a pure feature model does not clearly define which feature has to be executed prior to other ones. As such, we additionally annotate *Before Node* and *After Node* constraints if they cannot be derived based on input/output value dependencies.

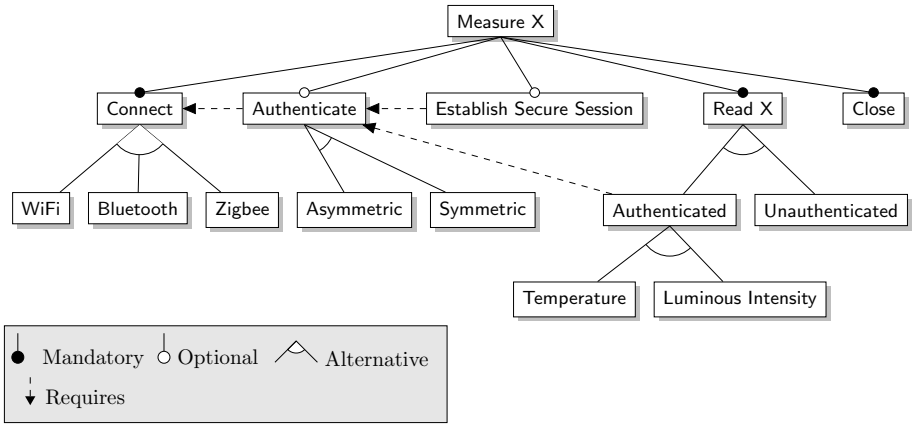

Fig. 5. Extract of the feature model that was used to derive the Microflow constraints of the Measure Temperature Microflow.

Consequently, the Microflow constraint model can be derived by applying simple model transformation rules. Thus, changes to the feature model are automatically propagated to the microservices and respectively to the microprocesses. This has the positive side effect that also development teams get a view on where and how their actual implementations are used within a business process. In addition, a strong connection is built between the process modeling team and the implementation team [2].

6 Conclusion

Today's industry is heavily driven by the digital transformation, short development cycles and highly flexible requirements from the market. We proposed a framework to use SPLE techniques for a combined variability modeling of BP models and software architectures. This leads to an integrated view of product variability from a business perspective as well as from a technical perspective.

In addition, we have shown how concepts of dynamic Microflow planning and execution allows workflow-based variability and WS orchestration during runtime. As illustrated in the case study, this is especially useful for industries providing service platforms such as IoT, where a number of service providers are publishing new WS. These WS can also make use of abstract microservices provided by the service platform to support access to the functionality of IoT devices in an abstract way, allowing third parties to develop microservices independent of the used infrastructure.

Future work will focus on extending the support for microservice orchestration, introducing advanced verification and validation techniques and a more fine-grained model to dynamically calculate path weights.

Acknowledgements. The project is funded by the Austrian Research Promotion Agency (FFG).

References

1. Xia, F., Yang, L.T., Wang, L., Vinel, A.: Internet of things. International Journal of Communication Systems **25**(9) (2012) 1101–1102
2. Sinnhofer, A.D., Pühringer, P., Oppermann, F.J., Potzmader, K., Orthacker, C., Steger, C., Kreiner, C.: Combining business process variability and software variability using traceable links. In Shishkov, B., ed.: Business Modeling and Software Design, Cham, Springer International Publishing (2018) 67–86
3. WFMC: Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels (1999)
4. Saidani, O., Nurcan, S.: Towards context aware business process modelling. In: 8th Workshop on Business Process Modeling, Development, and Support (BPMDS07), CAiSE. Volume 7. (2007) 1
5. Oberhauser, R., Stigler, S.: Microflows: Leveraging process mining and an automated constraint recommender for microflow modeling. In Shishkov, B., ed.: Business Modeling and Software Design, Cham, Springer International Publishing (2018) 25–48
6. Lewis, J., Fowler, M.: Microservices (2014)
7. Bouguettaya, A., Sheng, Q.Z., Daniel, F., eds.: Web Services Foundations. Springer (2014)
8. Sinnhofer, A.D., Pühringer, P., Potzmader, K., Orthacker, C., Steger, C., Kreiner, C.: Software configuration based on order processes. In Shishkov, B., ed.: Business Modeling and Software Design: 6th International Symposium, BMSD 2016, Rhodes, Greece, June 20–22, 2016, Revised Selected Papers, Cham, Springer International Publishing (2017) 200–220
9. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
10. Sinnhofer, A.D., Oppermann, F.J., Potzmader, K., Orthacker, C., Steger, C., Kreiner, C.: Increasing the visibility of requirements based on combined variability management. In Shishkov, B., ed.: Business Modeling and Software Design, Cham, Springer International Publishing (2018) 203–220
11. Sinnhofer, A.D., Pühringer, P., Kreiner, C.: varbpm — a product line for creating business process model variants. In: Proceedings of the Fifth International Symposium on Business Modeling and Software Design - Volume 1: BMSD 2015. (2015) 184–191
12. Oberhauser, R.: Microflows: Automated planning and enactment of dynamic workflows comprising semantically-annotated microservices. In Shishkov, B., ed.: Business Modeling and Software Design, Cham, Springer International Publishing (2017) 183–199
13. IBM: IBM business process manager 8.6.0 (2020)
14. Sheng, Q., Qiao, X., Vasilakos, A., Szabo, C., Bourne, S., Xu, X.: Web services composition: A decades overview. Information Sciences **280** (10 2014) 218238
15. Rajasekar, A., Wan, M., Moore, R., Schroeder, W.: Micro-services: A service-oriented paradigm for scalable, distributed data management. In: Data Intensive Distributed Computing: Challenges and Solutions for Large-scale Information Management, Hershey, PA, IGI Global (2012)

16. Rao, J., Su, X.: A survey of automated web service composition methods. In: Proceedings of the First International Conference on Semantic Web Services and Web Process Composition. SWSWPC04, Berlin, Heidelberg, Springer-Verlag (2004) 4354
17. Rosa, M.L., Aalst, W.M.P.V.D., Dumas, M., Milani, F.P.: Business process variability modeling: A survey. ACM Comput. Surv. **50**(1) (March 2017) 2:1–2:45
18. Derguech, W.: Towards a Framework for Business Process Models Reuse. In The CAiSE Doctoral Consortium (2010)
19. Gimenes, I., Fantinato, M., Toledo, M.: A Product Line for Business Process Management. Software Product Line Conference, International (2008) 265–274
20. Hallerbach, A., Bauer, T., Reichert, M.: Guaranteeing Soundness of Configurable Process Variants in Provop. In: Commerce and Enterprise Computing, 2009. CEC '09. IEEE Conference on, IEEE (2009) 98–105
21. Hallerbach, A., Bauer, T., Reichert, M.: Issues in modeling process variants with Provop. In Ardagna, D., Mecella, M., Yang, J., eds.: Business Process Management Workshops. Volume 17 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2009) 56–67
22. Reichert, M., Hallerbach, A., Bauer, T.: Lifecycle Support for Business Process Variants. In Jan vom Brocke and Michael Rosemann, ed.: Handbook on Business Process Management 1. Springer (2014)
23. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., La Rosa, M.: Configurable Workflow Models. International Journal of Cooperative Information Systems (2007)
24. La Rosa, M., Dumas, M., ter Hofstede, A.H.M., Mendling, J., Gottschalk, F.: Beyond control-flow: Extending business process configuration to roles and objects. In Li, Q., Spaccapietra, S., Yu, E., eds.: 27th International Conference on Conceptual Modeling (ER 2008), Barcelona, Spain, Springer (2008) 199–215
25. Haugen, O., Wasowski, A., Czarnecki, K.: Cvl: Common variability language. In: Proceedings of the 17th International Software Product Line Conference. SPLC '13 (2013)
26. Zhao, X., Zou, Y.: A business process-driven approach for generating software modules. Software: Practice and Experience **41**(10) (2011) 1049–1071
27. Sinnhofer, A.D., Pühringer, P., Potzmader, K., Orthacker, C., Steger, C., Kreiner, C.: A framework for process driven software configuration. In: Proceedings of the Sixth International Symposium on Business Modeling and Software Design - Volume 1: BMSD 2016. (2016) 196–203
28. Weiss, D.M., Lai, C.T.R.: Software Product-line Engineering: A Family-based Software Development Process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
29. Hammer, M., Champy, J.: Reengineering the Corporation - A Manifesto For Business Revolution. Harper Business (1993)
30. Österle, H.: Business Engineering - Prozess- und Systementwicklung. Springer-Verlag (1995)
31. OMG: Business process model and notation (bpmn) version 2.0 (2011)
32. Sinnhofer, A.D., Höller, A., Pühringer, P., Potzmader, K., Orthacker, C., Steger, C., Kreiner, C.: Combined variability management of business processes and software architectures. In: Proceedings of the Seventh International Symposium on Business Modeling and Software Design - Volume 1: BMSD,, INSTICC, SciTePress (2017) 36–45