Microflows: Leveraging Process Mining and an Automated Constraint Recommender for Microflow Modeling

Roy Oberhauser[0000-0002-7606-8226] and Sebastian Stigler

Computer Science Department, Aalen University, Aalen, Germany
{roy.oberhauser, sebastian.stigler}@hs-aalen.de

Abstract. Businesses and software development processes alike are being challenged by the digital transformation and agility trend. Business processes are increasingly being automated yet are also expected to be agile. Current business process modeling is typically labor-intensive and results in rigid process models. For larger processes it becomes arduous to consider all possible process variations and enactment circumstances. Contemporaneously, in software development microservices have become a popular software architectural style for partitioning business logic into fine-grained services accessible via lightweight protocols which can be rapidly and individually developed by small teams and flexibly (re)deployed. This results in an increasing number of available services and a much more dynamic IT service landscape. Thus, a more dynamic form of modeling, integration, and orchestration of these microservices with business processes is needed. This paper describes agile business process modeling with Microflows, an automatic lightweight declarative approach for the workflow-centric orchestration of semantically-annotated microservices using agent-based clients, graph-based methods, and the lightweight semantic vocabularies JSON-LD and Hydra. A graphical modeling tool supports Microflow modeling and provides dynamic constraint and microservice recommendations via a recommender service using machine learning of domain-categorized Microflows. To be able to utilize existing process model knowledge, a case study shows how Microflow constraints can be automatically extracted from existing Business Process Modeling Notation (BPMN) process files and transformed into flexible Microflow constraints, which can then be used to train the recommendation service. Further, it describes process mining of Microflow execution logs to automatically extract BPMN models and automated recovery for errors occurring during enactment.

Keywords: Business Process Modeling, Workflow Management Systems, Microservices, Service Orchestration, Agent Systems, Semantic Technology, Declarative Programming, Recommenders, Recommendation Engines, Business Process Mining, Business Process Modeling Notation.

1 Introduction

Congruent with and related to the digital transformation sweeping across businesses and industry, there is a growing emphasis on business agility and process automation. One key automation area are business processes (BP) or workflows, evidenced by $2.7 billion in spending on Business Process Management Systems (BPMS) (Gartner 2015).

The automation of a BP according to a set of procedural rules is known as a workflow (WF) (WfMC 1999). A workflow management system (WfMS), defines, creates, and manages the execution of workflows (WfMC 1999). However, with regard to agility, these workflows are often rigid, and while adaptive WfMS can handle certain adaptations, they usually involve manually intervention to determine the appropriate adaptation. Business Process Model and Notation (BPMN) (OMG 2011) supports Business process modeling (BPM) with a common notation standard. However, past BP models have hitherto not been accessible or leveraged in an automated way to support BPM.

To support digital automation, WFs often utilize software services. One trend supporting agility in the development and deployment of software is the now popular application of the microservice architecture style (Fowler and Lewis 2014). It provides an agile and loosely-coupled partitioning of business capabilities into fine-grained services individually evolvable, deployable, and accessible with lightweight mechanisms. However, as the dynamicity of the service world increases, the need for a more automated and dynamic approach to service orchestration becomes evident.

As the IT landscape becomes more complex and agile, it is evident that manual modeling will be handled by more automation. Approaches have included service orchestration, where a single executable process uses a flow description (such as WS-BPEL) to coordinate service interaction orchestrated from a single endpoint. In contrast, service choreography involves a decentralized collaborative interaction of services (Bouguettaya et al. 2014), while service composition involves the static or dynamic aggregation and binding of services into some abstract composite process. While automated dynamic workflow planning could potentially remove the manual overhead involved in workflow modeling, a fully automated semantic integration process remains challenging, with one study indicating that it is achieved by only 11% of Semantic Web applications (Heitmann et al. 2012).

Thus, in our view constraint-based declarative approaches toward process modeling provide maximum flexibility when searching the solution space for an optimal process model solution in an automated fashion. Thus, rather than pursue the heavyweight service-oriented architecture (SOA) and semantic web, we chose a pragmatic lightweight bottom-up approach. Analogous to the microservices principles, we use the term Microflow to mean lightweight workflow planning and enactment of microservices, i.e. a lightweight service orchestration of microservices. In our prior work, we described our declarative approach called Microflows for automatically planning and enacting lightweight dynamic workflows of semantically annotated microservices (Oberhauser 2017a) using cognitive agents and investigated its resource usage and viability (Oberhauser 2016). In (Oberhauser and Stigler 2017b), we extended our work to transform

existing BPMN models to Microflow constraints, as well as enabling bi-directional support for graphical modeling in BPMN tools via automated constraint extraction and BPMN generation from the execution log of a Microflow. Furthermore, automated error handing and on-the-fly replanning capabilities were extended to address the dynamic microservice landscape.

This paper contributes a graphical Microflow Modeler with drag-and-drop support that automatically generates the equivalent textual JSON Microflow constraints, provides a catalog of microservices currently available for a Microflow, and utilizes a Recommendation Service that suggests microservices or constraints based on prior Microflows. This can make Microflow modelers aware of constraints or microservices that occur frequently in Microflows within a certain domain. Note that the Microflow approach is not intended to address all facets of BPMS support, but focuses on a narrow area towards addressing the automatic orchestration of dynamic workflows given a multitude of microservices, and does so by using a pragmatic lightweight approach rather than a theoretical treatise.

This paper is organized as follows: the next section discusses related work. Section 3 presents the solution approach, while Section 4 describes its realization. The solution is evaluated in Section 5, which is followed by a conclusion.

2 Related Work

In IBM business process manager terminology microflow is used to mean a transient non-interruptible BPEL (Web Services Business Process Execution Language) process (IBM 2015), whereas in our terminology a microflow is independent of any BPMS, choreography, or orchestration language.

As to the combination of BPM with microservices, while Alpers et al. (2015) mention BPM with microservices, their focus is on collaborative BPM tool support services, presenting an architecture that groups them according to editor, management, analysis functionality, and presentation. Singer (2016) proposes a compiler-based actor-centric approach to directly compile Subject-oriented Business Process Management (S-BPM) models into a set of executable processes called microservices that coordinate work through the exchange of messages. In contrast, we assume our microservices preexist.

With regard to orchestration of microservices, related work includes Rajasekar et al. (2012), who describe the integrated Rule Oriented Data System (iRODS) for large-scale data management, which uses a distributed event-condition-action rule engine to orchestrate micro-services into conditional chain-oriented workflows, maintaining transactional properties through recovery micro-services. Alpers et al. (2015) describe a microservice architecture for BPM tools, highlighting a Petri Net editor to support humans with BPM. Sheng et al. (2014) surveys research prototypes and standards in the area of web service composition. Although the web service composition using the workflow technique (Rao and Su 2004) can be viewed as similar, our approach does not explicitly create an abstract composite service; rather, it can be viewed as automated dynamic web service orchestration using the workflow technique. Declarative approaches for process modeling include DECLARE (Pesic 2007). A DECLARE model

is mapped onto a set of LTL formulas that are used to automatically generate automata that support enactment. Adaptations with verification during enactment are supported, typically via GUI interaction with a human, whereby the changed model is reinitiated and its entire history replayed. As to inputs, DECLARE facilitates the definition of different constraint languages such as ConDec and DecSerFlow.

For combining multi-agent systems (MAS) and microservices, Florio (2015) proposes a MAS for decentralized self-adaptation of autonomous distributed components (Docker-based microservices) to address scalability, fault tolerance, and resource consumption. These agents known as selfLets mediate service decisions using partial knowledge and exchanging messages. Toffetti et al. (2015) provide a position paper focusing on microservice monitoring and proposing an architecture for scalable and resilient self-management of microservices by integrating management functions into the microservices, wherein service orchestration is cited to be an abstraction of deployment automation (Karagiannis et al. 2014), microservice composition or orchestration are not addressed.

Related standards include OWL-S (Semantic Markup for Web Services), an ontology of services for automatic web service discovery, invocation, and composition (Martin et al. 2004). Combining semantic technology with microservices, (Anderson et al. 2015) present an OWL-centric framework to create context-aware applications, integrating microservices to aggregate and process context information. For a more lightweight semantic description of microservices, JSON-LD (Lanthaler and Gütl 2012) and Hydra (Lanthaler 2013) (Lanthaler and Gütl 2013) provide a lightweight vocabulary for hypermedia-driven Web APIs and enable the creation of generic API clients.

Kluza et al. (2013) provide a survey of recommendation techniques for BPM and discuss machine learning (ML) approaches but do not address the use of neural networks and state that that feature extraction from BPMN diagrams is still an unsolved task. According to their classification we provide subject-based, position-based, and structural recommendations. As to BP recommenders, Chan et al. (2011) use a process fragment model and composition context graph with context matching to determine the requested service's behavior and implicitly infer the service's functionality to recommend a related or alternative service. They do not show an actual prototype user interface or a case study of it in use. Barba et al. (2013) propose a constraint-based BP recommendation system focused on planning and scheduling BP activities for performance goal optimization. Schobel and Reichert (2017) utilize machine learning to analyze historic BP performance, determine diverging processes, and recommend changes. Bobek et al. (2013) base recommendations on a Bayesian Network model that is built manually based on a configurable process. Our approach is to extract constraints and then use supervised learning via pre-classification by domain to automatically train an artificial neural multi-layer network to make recommendations.

In general, in contrast to the above work, our contribution specifically focuses on microservices with an automatic lightweight declarative approach for the workflow-centric orchestration of microservices using agent-based clients, graph-based methods, and lightweight semantic vocabularies like JSON-LD and Hydra. The extraction of goals and constraints from existing BPM in conjunction with modelling recommendations is supported, and error handling permits dynamic recovery and replanning.

3 Solution Approach

Referencing the solution architecture of Fig. 1, the principles and process constituting the solution approach are elucidated below and are based on Oberhauser (2016) and Oberhauser (2017a). One primary difference of our solution approach compared to typical BPM is the reliance on goal- and constraint-based agents using automated planners to navigate semantically-described microservices, thus the workflow is dynamically constructed, reducing the overall labor involved in manual modeling of rigid workflows that cannot automatically adapt to changes in the microservice landscape, analogous to the benefits of declarative over imperative programming.

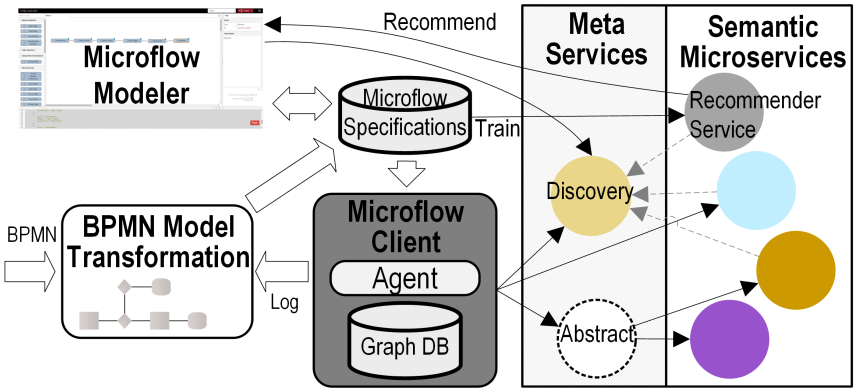

Fig. 1. Solution concept.

3.1 Microflow Principles

The solution approach consists of the following principles:

Microservice semantic self-description principle: Microservices provide sufficient semantic metadata to support autonomous client invocation, such that the client state at the point of invocation contains the semantic inputs required for the microservice invocation. Our realization uses JSON-LD/Hydra.

Client agent principle: For the client agent of Fig. 1, intelligent agents exhibit reactivity, proactiveness, and social ability, managing a model of their environment and can plan their actions and undertake goal-oriented behavior (Wooldridge 2009). Nominal WfMS are typically passive, executing a workflow according to a manually determined plan (workflow schema). Because of the expected scale in the number of possible microservices, the required goal-oriented choices in workflow modeling and planning, and the autonomous goal-directed action required during enactment, agent technology seems appropriate. Specifically, we chose Belief-Desire-Intention (BDI) agents (Bratman et al. 1988) for the client realization, providing belief (knowledge), desire via goals, and intention utilizing generated plans that are the workflow.

Graph of microservices principle: Microservices are mapped to nodes in a graph and can be stored in a graph database (see Fig. 1). Nodes in the graph are used to represent

any workflow activity, such as a microservice. Nodes are annotated with properties. Directed edges depict the directed connections (flows) between activities annotated via properties. To reduce redundant resource usage via multiple database instances, the graph database could be shared by the clients as an additional microservice.

Microflow as graph path principle: A directed graph of nodes corresponds to a workflow, a sequence of operations on those microservices, and is determined by an algorithm applied to the graph, such as shortest path. The enactment of the workflow involves the invocation of microservices, with inputs and outputs retained in the client and corresponding to the client state.

Declarative principle: Any workflow requirement specifications take the form of declarative goal and constraint modelling statements, such as the starting microservice type, end microservice type, and constraints such as sequencing or branch logic constraints. As shown under Models in Fig. 1, these specifications may be (automatically) extracted from an existing BPM should one exist, or (partially) discovered via process execution log mining.

Microservice discovery service principle (optional): We assume a microservice landscape to be much more dynamic with microservices coming and going in contrast to more heavyweight services. A microservice registry and discovery service (a type of Meta Service in Fig. 1) can be utilized to manage this and could be deployed in various ways, including centralized, distributed, client-embedded, with voluntary microservice-triggered registration or multicast-triggered mechanisms. For security purposes, there may be a desire to avoid discovery (of undocumented microservices) and thus maintain a whitelist. Clients thus may or may not have a priori knowledge of a particular microservice.

Abstract microservices principle (optional): Microservices with similar functionality (search, hotel booking, flight booking, etc.) can be grouped behind an abstract microservice (a type of Meta Service in Fig. 1). This simplifies constraints, allowing them to be based on a group rather than having to be individually based. It also provides an optional level of hierarchy to allow concrete microservices to only provide a client with a link to the logical next abstract microservice(s) without having to know the actual concrete ones, since the actual concrete microservice followers can be numerous and rapidly change, while determining exactly which ones are appropriate can perhaps best be decided by the client in conjunction with the abstract microservice.

Path weighting principle (optional): any follower of a service, be it abstract or concrete, can be weighted with a potentially dynamic cost that helps in quantifying and comparing one path with another in the form of relative cost. This also permits the navigation from one to another to be dynamically adjusted should that path incur issues such as frequent errors or slow responses. The planning agent can determine a minimal cost path.

Path constraint logic principle (optional): If the path weighting is insufficient and more complex logic is desired for assessing branching or error conditions, these can be provided in the form of constraints referencing scripts that contain the logic needed to determine the branch choice.

Note that the Data Repository and Graph Database could readily be shared as a common service, and need not be confined to the Client.

3.2 Microflow Lifecycle

Fig. 2. Microflow lifecycle.

The Microflow lifecycle involves five stages as shown in Fig. 2. In the *Microflow Modeling* stage, Microflow specifications (goal and constraints), retained as JSON files in a repository (Fig. 1), are modeled: 1) graphically (in our Microflow Modeler and Constraint Recommender), 2) textually, 3) extracted and transformed via tools from existing BPMN process models, or 4) via process mining of execution logs (e.g., Microflow logs) and transformed to BPMN and then Microflows.

The *Microservice Discovery* stage involves utilizing a microservice discovery service to build a graph of nodes containing the properties of the microservices and links (followers) to other microservices, analogous to mapping the landscape.

In the *Microflow Planning* stage, an agent takes the goal and other constraints and creates a plan known as a Microflow, finding an appropriate start and end node and using an algorithm such as shortest path to determine a directed path. In our opinion, a completely dynamic enactment without any planning (no schema) could readily lead to dead-end or circular paths, causing a waste of unnecessary invocations that do not lead to the desired goal and can potentially not be undone. This is analogous to following hyperlinks without a plan, which does not lead to the goal and require backtracking. Alternatively, replanning after each microservice invocation involves planning resource overhead (CPU, memory, network), and since this is unlikely to dynamically change within the enactment lifecycle, we chose a pragmatic and lightweight approach from a resource utilization perspective: plan once and then enact until an exception occurs, at which point a necessary replanning is triggered. Further advantages of our approach, in contrast to a thoroughly adhoc approach, is that the client is assured that there is at least one path to the goal before starting, and validation of various structural, semantic, and syntactic aspects can be readily performed.

In the *Microflow Enactment* stage, the Microflow is executed by invoking each microservice in the order of the plan, typically sequentially but it could involve parallel invocations. A replanning of the remaining Microflow can be performed if an exception occurs or if notified by the discovery service of changes to the set of microservices. A client should retain the Microflow model (plan) and be able to utilize the service interfaces and thus have sufficient semantic knowledge for enactment.

The *Microflow Analysis* stage involves the monitoring, analysis, and mining of execution logs in order to improve future planning. This could be local, in a trusted environment, or this could be distributed. Thus, if invocation of a microservice has often resulted in exceptions, future planning for this client or other clients could avoid this troublesome microservice. Furthermore, the actual latency incurred for usage of a microservice could be tracked and shared between agents and taken into account as a type of cost in the graph algorithm.

4 Realization

As various details of our Microflow realization and lifecycle were previously detailed in (Oberhauser, 2016) and (Oberhauser, 2017), a short summary is provided and the rest of this section details the newer extensions.

Implementations of microservices are assumed to be REST compliant using JSON-LD and Hydra descriptions. For our prototype testing, REST (REpresentational State Transfer) and HATEOAS support (Fielding 2000) were integrated with Spring-boot-starter-web v. 1.2.4, which includes Spring boot 1.2.4, Spring-core and Spring-web v. 4.1.6, Embedded Tomcat v. 8.0.23; Hydra-spring v. 0.2.0-beta3; and Spring-hateoas v. 0.16 are integrated. For JSON (de)serialization Gson v. 2.6.1 is used. Unirest v. 1.3.0 is used to send HTTP requests. As a REST-based discovery service, Netflix's open source Eureka v. 1.1.147 is used.

The Microflow clients uses the BDI agent framework Jadex v. 3.0-SNAPSHOT (Pokahr et al. 2005). Jadex's BDI nomenclature consists of Goals (Desires), Plans (Intentions), and Beliefs. Beliefs can be represented by attributes like lists and maps. Three agents were created: the DataAgent is responsible for providing for and maintaining data repository, the PlanningAgent generates a path through the graph as a Microflow, while the ExecutionAgent communicates directly with microservices to invoke them according to the Microflow. Neo4j and Neo4j-Server v. 2.3.2 is used as a client Data Repository.

Microflow specifications (goals and constraints) are referred to as PathParameters and consist of the startServiceType, endServiceType, and constraint tuples. Each constraint tuple consists of the target of the constraint (the service type affected), the constraint, and a constraint type (required, beforeNode, afterNode). For instance, target = "Book Hotel", constraint = "Search Hotel", and constraint type = "afterNode" would be read as: "BookHotel" is after node "Search Hotel", implying the Microflow sequencing must ensure that "Search Hotel" precedes "Book Hotel" (but does not require that it must be directly before it).

During Microflow Planning, constraint tuples are analyzed, whereby any AfterNode is converted to a BeforeNode by swapping target and constraint, RequiredNode constraints are also converted to BeforeNode constraints, and redundant constraints are removed and the constraints are then ordered.

4.1 BPMN to Microflow Specification Transformation

To leverage existing process model knowledge, a BPMN-to-Microflow (B2M) transformation tool is implemented in Java and parses BPMN 2.0 files, automatically extracting the start and end node (goal) and any constraints, generating a Microflow JSON specification file (left corner of Fig. 1) for the Microflow repository. The java libraries camunda-bpmn-model and camunda-xml-model version 7.6.0 are utilized for parsing. It includes support for the following BPMN elements: activities, events, gateways, and connections. Currently unsupported in the implementation for automated extraction are swimlanes, artifacts, and event subprocesses (throwing, catching, and interrupting events). Some of these can be manually modelled in the Microflows using scripting.

4.2 Microflow Constraint Mining

A MicroflowLog-BPMN mining tool is implemented in Java that automatically parses a Microflow execution log file (taking the Log in Fig. 1 as input) and generates a BPMN 2.0 file, which could in turn be automatically converted to a Microflow specification file, for instance if constraint extraction is desired. Since it generates a direct sequence of the actual path taken, it results in a simple sequence of tasks. However, this can be helpful in providing a graphical depiction for human analysis and comparison, determining issues, debugging constraints, and as a reference or starting point for models having greater complexity.

4.3 Recommender Service

A Microflow modeler is confronted with many options and decisions, the number of possible Microservices to consider can be quite large, and there are various constraints that may or may not be appropriate, yet if certain essential constraints or Microservices are missing the Microflow may be problematic. To assist the modeler and raise awareness of possibly relevant constraints as well as microservices, a Microflow Recommender Service (shown in Fig. 1) utilizing machine learning (ML) was realized.

It creates an artificial neural multi-layer network using DeepLearning4J (DL4J) 0.7.2 and provides a global recommendation of microservice types frequently used together with an input type, as well as frequent before and after constraints for some selected microservice. Three layers were used: Layer 0 takes the different constraints as input and has 200 fixed outputs, Layer 1 has both 200 inputs and 200 outputs, and Layer 2 takes 200 inputs and produces a probability distribution of available constraints as output equivalent to the number of inputs. Other relevant parameters are: the layer size was set to 200, total number of training epochs was set to 100, 8 constraints are batched at once for training, the stochastic gradient descent optimization, learning rate of 0.1 as to what degree correct suggestions should improve certainty, rms decay of 0.95 as to what degree unused suggestions lose relevance, truncated backpropagation, with further parameters left to their default.

Supervised learning using classification (pre-classification by domain) is used to train the network from a repository of Microflow specifications, one network per domain. All Microflows constraints within a domain are aggregated, and during training the ordering of available constraint inputs is randomized. To provide a recommendation, a single time step is run to feed the current constraints through the network, and then the network output is converted into a JSON constraint recommendation.

The REST interface for the service (DELETE omitted) consists of:

`GET /domains` – returns list of available domains and POST and DELETE to add and remove specific domains respectively.

`POST /recommendation/{domain}` - based on an input of a Microflow as JSON and given a selected node, returns a recommendation consisting of a list of global microservice types, a list of predecessor constraints, and a list of follower constraints.

`POST /train/{domain}` – trains system with an additional list of constraints

`GET /train/{domain}` – returns the list of constraints used to train the system

4.4 Microflow Modeler and the Recommender Service

The Microflow Modeler Tool provides a graphical and text-based editor to assist with the Microflow modeling process (see Fig. 3). It is implemented using NodeRED, a web-based tool for flow-based programming. Nodes or building blocks can connected to each other to direct the information flow or execute functionality. A Microflow constraint is in essence a description of a relation between microservices and can be depicted via a connection.

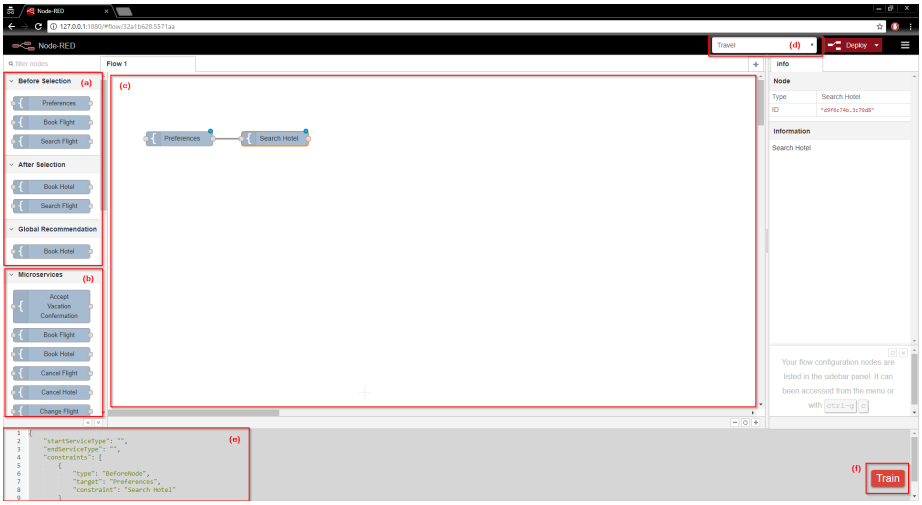

Fig. 3. Microflow Modeler Tool.

Fig. 3c shows the Microflow modelling workspace, where nodes representing Microservices can be inserted and moved and relations drawn to indicate before and after constraints. Fig. 3e shows the generated textual equivalent of the graphical model on the workspace, and can be directly edited with changes dynamically reflected in the graphical model (round-trip). Fig. 3b shows Microservices, a catalog of the available microservices, any of which can be pulled into the workspace via drag-and-drop. Fig. 3d shows a drop-down menu to select the modeling domain (e.g., Travel, Business, Health). This controls the domain (category) of the Microflow repository used for constraining recommendations and categorizing this Microflow. Nearby is the Deploy button, which saves the Microservice specification to the repository and permits execution. that includes recommendations based on mining historical Microflows and, utilizing machine learning, recommends microservices and constraints included in past workflows (BPMN or Microflows). Any change to the constraints causes an update request for recommendations. Fig. 3a shows the recommendation area, which includes a Global Recommendation for a suggested microservice that was most frequently included in this domain or based on the currently selected microservice type. Fig. 3f starts a Recommender Service training session with the given Microflow. The background processing performed in the Microflow Modeler is shown in Fig. 4.

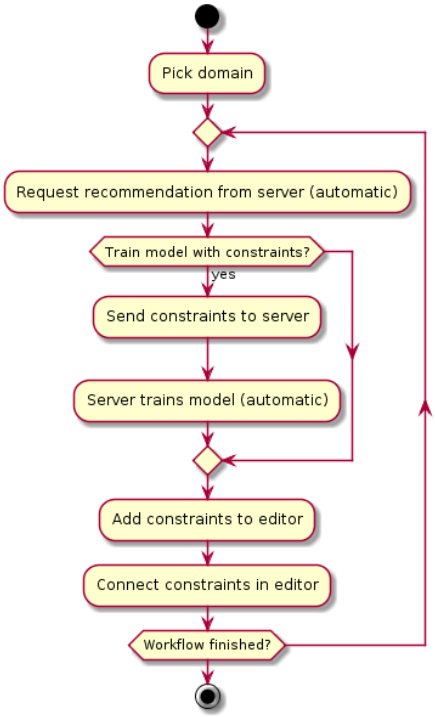

Fig. 4. Microflow Modeler background processing.

4.5 Microflow Error Recovery

To support enactment error recovery, the Microflow client now supports data versioning of its state, integrating the javersion data versioning toolkit v. 0.14. The algorithm is shown in Fig. 5 and referred to by line. At each abstract node, the current client state (JSON data outputs from microservices) is committed (Line 11). If the execution of a microservice is not successful, the transition is penalized by adding to its cost so that any replanning does not necessarily continue to include a microservice with constant issues (Line 22); the node index is set to the last node where a commit was performed (Line 24) (ultimately the start node if none) and its state at that node restored (analogous to a rollback); and a replanning is initiated (Line 25) from that node.

Thus, Microflow clients support an automated recovery and replanning mechanism. This is in contrast to standard BPMS whereby an unhandled exception typically results in the process terminating. In contrast to basic HATEOAS client implementations, the client state can be rolled back to the last known good service and a replanning enables the client to seek an alternative to reach its goal. This error recovery technique can be used to support the Microflow equivalent of BPMN subprocess transactions.

```
1   procedure ExecuteWorkflow(path, constraints, initialInput)
2    lastAbstractNode ← null
3    availableInput ← MapOfListOfStings()
4    availableInput.add("START", initialInput)
5    nodeIndex ← 0
6    while nodeIndex < length(path) do
7     description ← getNodeInfoForNodeInPath(nodeIndex, path)
8     inputs ← availableInput.getAllValidInputsFor(description)
9     if isAbstactNode(description) then
10     if lastAbstractNode != description then
11      revisionID ← commit(nodeIndex, availableInput)
12      enqueue(revisionID)
13      lastAbstractNode ← description
14     end if
15    else   // current node is a regular microservice
16     state ← NONE
17     for each input in inputs do
18      state, result ← executeMicroservice(description, input)
19      if state == SUCCESS then
20       availableInput.add(description, result)
21      else  // state == Exception
22       penalizeTransitionToCurrentNode(nodeIndex)
23       revisionID ← dequeue()
24       resetNodeIndex, availableInput ← checkout(revisionID)
25       newPath  ← getNewShortestPathAfterException(path, resetNodeIndex, constraints)
26       nodeIndex ← resetNodeIndex
27       path ← newPath
28       break
29      end if
30     end for
31     if state == SUCCESS then
32      if isBranchingNode(description,constraints) then
33       newNodeIndex ← runGroovyScriptFromConstraint(description,constraints)
34       newPath ← getNewShortestPathForAnotherBranch(path, nodeIndex, newNodeIndex)
35       nodeIndex ← newNodeIndex
36       path ← newPath
37      else   // current node is not a branching one
38       nodeIndex++
39      end if
40     end if
41    end if
42   end while
43   return availableInput
44  end procedure
```

Fig. 5. Microflow execution algorithm.

5 EVALUATION

Case studies is used to evaluate the solution, first considering the extraction of constraints from BPMN models, the mining of BPMN models from a Microflow execution log, usage of the Microflow Modeler and then error recovery.

5.1 BPMN Transformation

As an illustrative example, we created our own travel booking process shown in Fig. 6, whereby both a hotel and flight should be found, and then a booking (reservation) of each is performed, and then payment is collected. Virtual microservices are used during enactment that differentiate themselves semantically but provide no real invocation functionality. The BPMN model of Fig. 6 generated an XML file using Camunda Modeler consisting of 209 lines and 11372 characters. In contrast, the Microflow constraint JSON file generated from this model by our BPMN-Microflow transformation tool contains 14 lines and 460 characters (Fig. 9a).

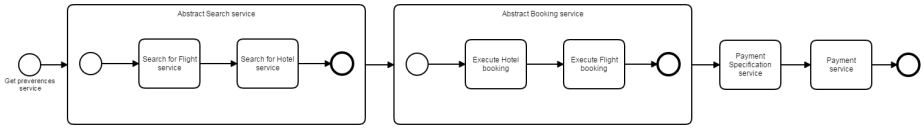

Fig. 6. Our travel booking example as BPMN.

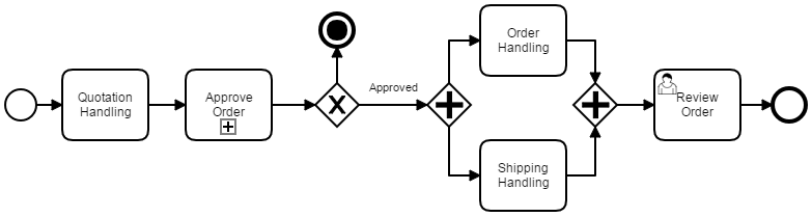

Fig. 7. Collapsed SubProcess BPMN model from OMG (2010).

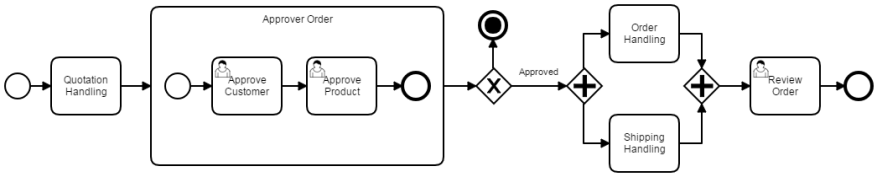

Fig. 8. Expanded SubProcess BPMN model from OMG (2010).


```
{ "startServiceType":"Preferences",
  "endServiceType":"Payment",
  "constraints":[
    { "type":"RequiredNode",
      "target":"Flight Search"},
    { "type":"AfterNode",
      "target":"Payment",
      "constraint":"Book Hotel"},
    { "type":"BeforeNode",
      "target":"Hotel Search",
      "constraint":"Book Hotel"},
    { "type":"AfterNode",
      "target":"Payment",
      "constraint":"Book Flight" }]}
```


(a)


```
{ "startServiceType":"StartProcess",
"endServiceType":"EndProcess",
"terminateServiceType":"TerminateProcess",
"constraints":[
 { "type":"BeforeNode",
   "target":"Shipping Handling",
   "constraint":"Review Order"},
 { "type":"BeforeNode",
   "target":"Order  Handling",
   "constraint":"Review Order"},
 { "type":"BranchAfterExecution",
   "target":"Approve Product",
   "constraint":"TerminateOrContinue.groovy"},
 { "type":"BeforeNode",
   "target":"Approve Customer",
   "constraint":"Approve Product"},
 { "type":"BeforeNode",
   "target":"Quotation Handling",
   "constraint":"Approve Customer"}]}
```


(b)

Fig. 9. Extracted constraints from the BPMN of (a) Travel booking and (b) SubProcess examples.

To determine to what extent the spectrum of BPMN 2.0 is supported and if any issues are a result of the approach or limitations of the implementation, the BPMN files from

OMG BPMN Examples (OMG 2010) were tested. Both the collapsed SubProcess (Fig. 7) as well as the Expanded SubProcess (Fig. 8) BPMN models consist of 222 lines and 13996 characters of BPMN XML and were automatically transformed to constraint files of 19 lines and 622 characters in Microflow JSON as shown in Fig. 9b. Both BPMN files contain the subprocess information which is hidden in the graphical representation in Fig. 8.

Assessing the subset of BPMN transformations of the OMG BPMN examples that were unsuccessful, which included portions of Incident Management, Nobel Prize Process, Procurement Process with Error Handling, Travel Booking, Pizza Order Process, Hardware Retailer, Email Voting, we identified the following issues:

- Multiple start events: this implies multiple processes are enacted concurrently, resulting in issues with planning and merging state and potential race conditions. These issues, however, are due to limitations with our prototype implementation, not of the approach. Future work will consider concurrent enactment and synchronization.
- Multiple end or terminate events: in this case, the planner cannot identify the goal node for the Microflow. One current implementation workaround is to create an abstract final node or a final common end node, which can be inserted into our internal graph with the appropriate additional relations.
- Missing start and end events: these are optional in BPMN and result in no clear start and end goal for the planner. One workaround for our implementation is to assume these are implied based on activities having no predecessor or no successor.
- Event subprocess: the prototype does not automatically map exception areas, yet it would be feasible by adding a constraint to each contained node with a conditional before whereby a new path is then dynamically replanned from this relation on error.
- Swim lanes: currently only isolated swim lanes are supported, but future work will consider a mapping to abstract nodes and possible communication and synchronization support.
- Artifacts: our implementation cannot map BPMN inputs since in these models they lack sufficient semantic detail. One workaround would be to provide a manually created map of BPMN types to JSON-LD types.

5.2 Microflow Constraint Mining

From a Microflow execution log (Fig. 10a) that injected an automated error recovery condition for the Travel example of Figs. 6 and 9a, our MicroflowLog-BPMN mining tool extracted a BPMN file (Fig. 10c shows an excerpt from its BPMN XML file and Fig. 10d its graphical equivalent). As explained in Section 4.2, this can assist human analysis or serve as a starting point for further modeling. To demonstrate the feasibility of a full cycle (roundtrip) back to a Microflow specification from an execution log, this BPMN was transformed to Microflow constraints shown in Fig. 10b. These constraints could, for example, then be reduced by a human to only those truly required and adjusted for requisite sequencing in order to optimize the dynamic planning capability.

Fig. 10. Travel Booking example (a) Microflow process log file output with recovery elements highlighted in bold; (b) extracted Microflow constraints; (c) extracted BPMN XML; and (d) BPMN graphical equivalent.

5.3 Microflow Modeler and Recommender Service Case Study

A case study is used to demonstrate Microflow Modeler and Constraint Recommender capabilities.

5.3.1 Recommender Service Training Set

In searching for available BPMN diagrams for testing we faced various difficulties. The OMG (2010) BPMN examples lack significant variants for a domain. Few companies are willing to share their internal processes for various reasons and potential risks, as these are often seen as a competitive advantage and significant investments in business process modeling were made. Ones we did publically available find were difficult to categorize (service names vary tremendously between organizations and no obvious semantic equivalence was available based on the service name itself) and they also lacked sufficient variation within a domain. Specifically, we were looking for reoccurring services use in various workflows. We thus chose to develop a synthetic Microflow repository dataset to ensure that certain reoccurring sequences that map to constraints would occur and to use semantically equivalent service names within three different domains. These are fairly basic Microflows, and consisted of ten travel Microflows, ten health Microflows, and eight business Microflows, samples of which are shown in Figs. 11, 12, and 13 respectively. This is not ideal for training ML, and one would wish there were large accessible repositories with clarity on the semantic mapping between services. Nevertheless, they provide an initial starting point for the evaluation.

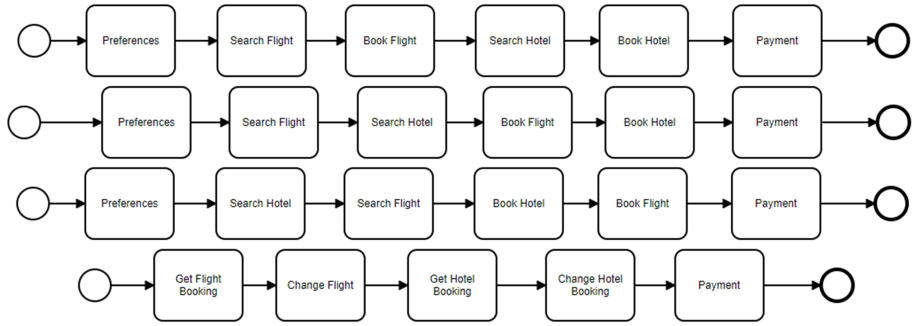

Fig. 11. Sample of BPMN travel domain training workflows variants.

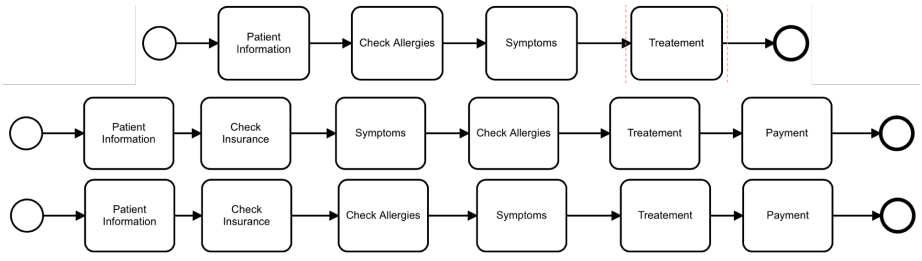

Fig. 12. Sample of BPMN health domain training workflows variants.

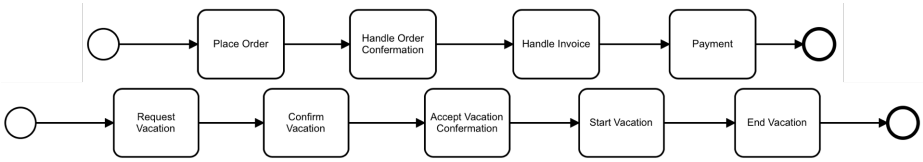

Fig. 13. Sample of BPMN business domain training workflows variants.

For instance, in the travel domain the recurring pattern is Preferences before other services, and payment after any other services. Likewise, Search occurs before Book. In the health domain the pattern is that Patient Information comes first, and Treatment comes only after checking various other information. Thus, domain-specific knowledge in the form of constraints are now made readily available for ML training without an expert being available by automatically extracting constraints from available process knowledge held within the process models.

5.3.2 Recommender Service Usage

In the following case study, the Recommender Service is used in conjunction the Microflow Modeler after having been trained with the Microflow specifications mentioned in Section 5.3.1. Fig. 14 shows the initial state with the domain Travel selected. A global recommendation to include the Book Flight microservice is made on the basis of the frequency of occurrence of this microservice in the models in this domain.

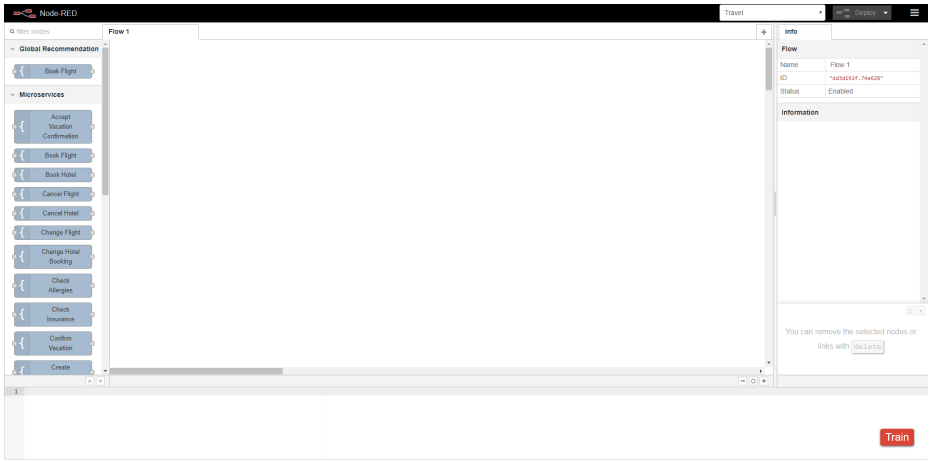

Fig. 14. Microflow Modeler initial state for the travel domain.

In Fig. 15a, a Preferences microservices was added and is currently selected (outlined in red). Recommendations for its followers ‘After Selection’ are ‘Search Hotel’ and ‘Search Flight’ with a global recommendation to include ‘Book Hotel’ somewhere in

the Microflow. In Fig. 15b, ‘Search Hotel’ was dragged and dropped behind Preferences and connected as its follower. No microservice is selected so the Before and After are irrelevant. Book Hotel is still recommended, but ‘Search Flight’ was added after ‘Search Hotel’ in Fig. 15c. In Fig. 16, the Microflow has been further modeled to include Payment, which is currently selected. No ‘After’ microservice is recommended (because no microservices in the training came after this), for Before recommendations ‘Cancel Hotel’ and ‘Change Hotel Booking’ are shown, and also ‘Book Flight’ since it could come directly before Payment. ‘Change Flight’ is now a global recommendation.

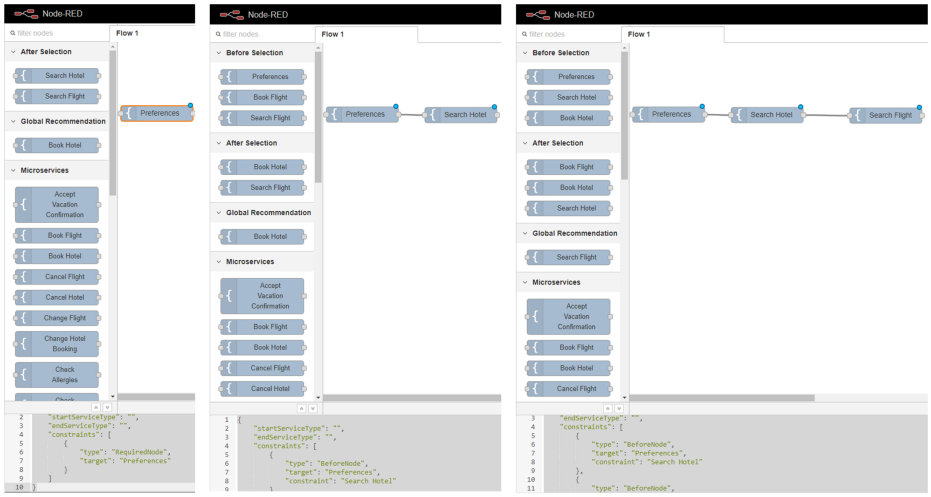

Fig. 15. Microservices with constraints added to a Microflow in the travel domain.

Fig. 16. Microflow Modeler showing a completed Microflow in the travel domain.

Fig. 17 shows that the recommendations differ when the selected domain is health care, for the selected node ‘Check ‘Insurance’ a Before suggestion is ‘Patient Information’ and After suggestions are Payment and Symptoms, whereby Treatment is given as a global recommendation to be included somewhere. Fig. 18 shows a business domain example where ‘Confirm Vacation’ is selected, with ‘Accept Vacation Confirmation’ suggested under After and this happens to also be the top global recommendation.

Fig. 17. Sample of BPMN business domain training workflows variants.

Fig. 18. Sample of BPMN business domain training workflows variants.

5.3.3 Recommender Service Technical Evaluation

Performance measurements were made to assess the practicality of usage on typical PCs. For the measurements, the hardware configuration consisted of an i5-4460@ 3,20 GHz and 8GB of RAM. The software configuration was Win10 Pro, Java 1.8.0_144, NodeRed based on 0.17.5, and DeepLearning4J 0.7.2.

Training the system took 22.4 seconds on average (over five invocations). The initial get recommendation request involves initialization and took 221 milliseconds total. Thereafter, any recommendation requests took 6.9 milliseconds on average (over five invocations).

Based on these results, the system seems fluid and viable for dynamic modeling and use of recommendations. While the training time has a noticeable delay (which is also why we have an explicit button in red), this is likely not to occur as frequently in usage and could be done without hampering modeling work in the background or at night, or the microservice could be placed on a high-performance server. No explicit tuning was performed.

5.4 Microflow Error Recovery

To demonstrate the automated error recovery capability, the Flight Booking service was modified to return a HTTP 500 status code and a Recovery for Flight Booking micro-service (which could for example attempt to restart the failing service) was added as a microservice with a path cost higher than that of the normal Flight Booking just to demonstrate the ability for replanning to adjust and take a different path after receiving an error. It does not imply that recovery microservices are needed.

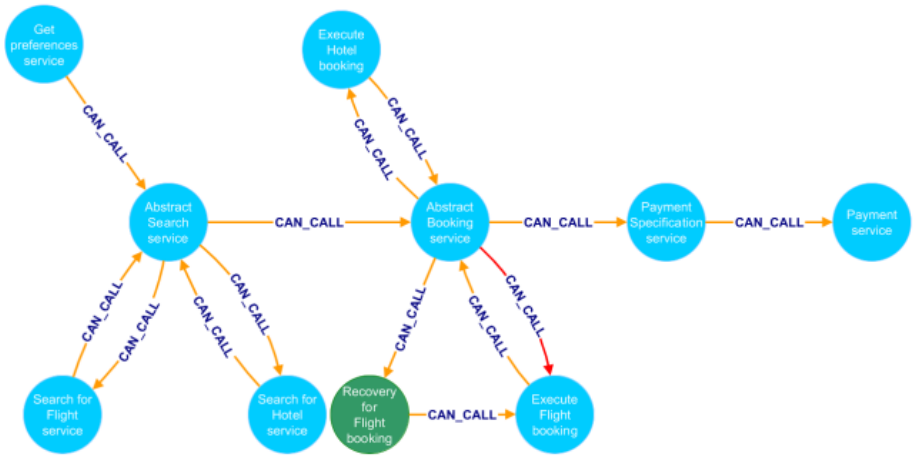

Fig. 19. Travel Booking example as Neo4J graph (error recovery shown in green).

Fig. 20. Output of client state in JSON.

Fig. 19 includes a recovery microservice (green). In the execution log file of Fig. 10a, after receiving an error the execution returns to Abstract Booking Service. The client state (shown in Fig. 20) is restored to that which it was at the last commit, leaving ItemList, Hotel, and Flight (Lines 5-10) and discarding LodgingReservation and Flight-Reservation (Lines 1-4). The relation between Abstract Booking and Flight Booking is penalized, resulting in a replanning from Abstract Booking that now includes Recovery for Flight Booking since it is the path with the least cost. This is seen in Fig. 10a with the difference in the planning sequence from [CAN_CALL,9] to [CAN_CALL,12]-->(10)-->[CAN_CALL,13].

6 Conclusion

In this paper, we described business process mining for constraints, Microflow modeling tool support, constraint recommendations based on machine learning, automatic lightweight declarative workflow-based orchestration of semantically-annotated microservices using agent-based clients, graph-based methods, and lightweight semantic vocabularies. The solution principles of the Microflow approach and its lifecycle were elucidated and details on its realization. The evaluation showed that Microflow constraints can be automatically extracted from existing BPMN files, that Microflow execution log file process mining can be used to extract BPMN models, that these can be used to train a constraint recommender service using machine learning, and that certain types of client error recovery can be automated with client state rollback, path cost penalization, and dynamic replanning during enactment. The Microflow constraint specification files were found to be much smaller than the equivalent BPMN files.

With the Microflow approach, only the essential rigidity is specified via constraints, permitting a greater degree of agility in the business process models since the remaining unspecified areas of the workflow are automatically determined and planned (and thus remain dynamically adaptable). This significantly reduces business process modeling labor and permits a higher degree of reuse in a dynamic microservice world, reducing the total cost of ownership. Since the workflow (or plan) is not completely adhoc and dynamic, validation and verification checks can be performed before execution begins, and one is assured that the workflow is executable as planned. However, enhanced support for verification and validation of the correctness of the Microflow is still required for users to entrust the automatic planning. By integrating Microflow constraint recommendations in the Microflow Modeler, modeling mistakes due to lack of awareness or forgetfulness can be reduced as the constraint complexity increases.

Future work includes expanded support for BPMN 2.0 elements in our implementation, integrating advanced verification and validation techniques, integrating semantic support in the discovery service, supporting compensation and long-running processes, enhancing the declarative and semantic support and capabilities, tuning the recommender service, and empirical and industrial usage studies.

Acknowledgments. The authors thank Florian Sorg and Tobias Maas for their assistance with the design, implementation, evaluation, and diagrams.

7 References

Alpers, S., Becker, C., Oberweis, A., Schuster, T.: Microservice based tool support for business process modelling. In: IEEE 19th International Enterprise Distributed Object Computing Workshop (EDOCW), pp. 71-78. IEEE (2015).

Anderson, C., Suarez, I., Xu, Y., David, K.: An Ontology-Based Reasoning Framework for Context-Aware Applications. In: Modeling and Using Context, pp. 471-476. Springer International Publishing (2015).

Barba, I., Weber, B., Del Valle, C., Jiménez-Ramírez, A.: User recommendations for the optimized execution of business processes. Data & Knowledge Engineering, 86, 61-84 (2013).

Bobek, S., Baran, M., Kluza, K., Nalepa, G.J.: Application of Bayesian Networks to Recommendations in Business Process Modeling. In: Proceedings of the Workshop AI Meets Business Processes 2013, pp. 41-50. ceur-ws.org (2013).

Bouguettaya, A., Sheng, Q., Daniel, F.: Web services foundations. Springer (2014).

Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational intelligence, 4(3), 349-355 (1988).

Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine (2000).

Florio, L.: Decentralized self-adaptation in large-scale distributed systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 1022-1025. ACM (2015).

Fowler, M., Lewis, J.: Microservices a definition of this new architectural term, http://martinfowler.com/articles/microservices.htm, last accessed 2018/1/31

Gartner: Gartner Says Spending on Business Process Management Suites to Reach $2.7 Billion in 2015 as Organizations Digitalize Processes (Press release), https://www.gartner.com/newsroom/id/3064717, last accessed 2018/1/31

Heitmann, B., Cyganiak, R., Hayes, C., Decker, S.: An empirically grounded conceptual architecture for applications on the web of data. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 42(1), 51-60 (2012).

IBM: IBM Business Process Manager V8.5.6 documentation, http://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.6/com.ibm.wbpm.wid.bpel.doc/topics/cprocess_transaction_micro.html, last accessed 2018/1/31

Karagiannis, G. et al.: Mobile cloud networking: Virtualisation of cellular networks. In: 21st International Conference on Telecommunications (ICT), pp. 410-415. IEEE (2014).

Kluza, K., Baran, M., Bobek, S., Nalepa, G. J.: Overview of recommendation techniques in business process modeling. In: Proceedings of 9th Workshop on Knowledge Engineering and Software Engineering (KESE9), pp. 46-57. CEUR-WS.org (2013).

Lanthaler, M.: Creating 3rd generation web APIs with hydra. In: Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion), pp. 35-38. ACM, New York, NY, USA, (2013). DOI: http://dx.doi.org/10.1145/2487788.2487799

Lanthaler, M., Gütl, C.: On using JSON-LD to create evolvable RESTful services. In: Proceedings of the Third International Workshop on RESTful Design, pp. 25-32. ACM (2012).

Lanthaler, M., Gütl, C.: Hydra: A Vocabulary for Hypermedia-Driven Web APIs. In: Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013) at the 22nd International World Wide Web Conference (WWW2013), vol. 996. CEUR-WS (2013).

Martin, D. et al.: OWL-S: Semantic markup for web services. W3C member submission, W3C (2004).

OMG: BPMN 2.0 by Example Version 1.0. OMG (2010).

OMG: Business Process Model and Notation (BPMN) Version 2.0. OMG (2011).

Oberhauser, R.: Microflows: Lightweight Automated Planning and Enactment of Workflows Comprising Semantically-Annotated Microservices. In: Proceedings of the Sixth International Symposium on Business Modeling and Software Design (BMSD 2016), pp. 134-143. SCITEPRESS (2016).

Oberhauser, R.: Microflows: Automated Planning and Enactment of Dynamic Workflows Comprising Semantically-Annotated Microservices. In: 6th International Symposium on Business Modeling and Software Design (BMSD 2016), Revised Selected Papers, B. Shishkov (Ed.). LNBIP, Vol. 275, pp. 183-199. Springer International Publishing (2017a).

Oberhauser, R., Stigler, S.: Microflows: Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices. In: Proceedings of the Seventh International Symposium on Business Modeling and Software Design (BMSD 2017), pp. 19-28. SCITEPRESS (2017b).

Pesic, M., Schonenberg, H., van der Aalst, W. M.: Declare: Full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 287-287. IEEE (2007).

Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In: Multi-agent programming, pp. 149-174. Springer (2005).

Rajasekar, A., Wan, M., Moore, R., Schroeder, W.: Micro-Services: A Service-Oriented Paradigm for Data Intensive Distributed Computing. In: Challenges and Solutions for Large-scale Information Management, pp. 74-93. IGI Global (2012).

Rao, J., Su, X.: A survey of automated web service composition methods. In: Semantic Web Services and Web Process Composition, pp. 43-54. Springer Berlin Heidelberg (2004).

Schobel, J., Reichert, M.: A Predictive Approach Enabling Process Execution Recommendations. In Advances in Intelligent Process-Aware Information Systems, pp. 155-170. Springer, Cham (2017).

Sheng, Q. Z. et al.: Web services composition: A decade's overview. Information Sciences, 280, 218-238 (2014).

Singer, R.: Agent-Based Business Process Modeling and Execution: Steps Towards a Compiler-Virtual Machine Architecture. In: Proceedings of the 8th International Conference on Subject-oriented Business Process Management. ACM (2016).

Toffetti, G., Brunner, S., Blöchlinger, M., Dudouet, F., Edmonds, A.: An architecture for self-managing microservices. In: Proceedings of the 1st International Workshop on Automated Incident Management in Cloud, pp. 19-24. ACM (2015).

WfMC: Workflow Management Coalition Terminology & Glossary, WFMC-TC-1011, Issue 3.0. Workflow Management Coalition (1999).

Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley & Sons (2009).