# Microflows: Enabling Agile Business Process Modeling to Orchestrate Semantically-Annotated Microservices

Roy Oberhauser and Sebastian Stigler
*Computer Science Department, Aalen University, Aalen, Germany*
*{roy.oberhauser, sebastian.stigler}@hs-aalen.de*

Keywords: Business Process Modeling, Workflow Management Systems, Microservices, Service Orchestration, Agent Systems, Semantic Technology, Declarative Programming.

Abstract: Businesses and software development processes alike are being challenged by the digital transformation trend. Business processes are increasingly being automated yet are expected to be agile. Current business process modeling is typically labor-intensive and results in rigid process models, with larger process models unable to cope with all possible process variations and enactment circumstances. In software development, microservices have become a popular software architectural style for partitioning business logic into fine-grained services that can be rapidly and individually developed and (re)deployed while accessed via lightweight protocols, resulting in many more services and a much more dynamic service landscape. Thus, a more dynamic form of modeling, integration, and orchestration of 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 case study shows how Microflow constraints can be automatically extracted from existing Business Process Modeling Notation (BPMN) files, how Microflow execution log file process mining can be used to extract BPMN models, and demonstrates an automated error recovery capability during enactment.

## 1 INTRODUCTION

The digital transformation sweeping through society affects businesses everywhere, resulting in an increased emphasis on business agility and automation. Business processes or workflows are one primary automation area, evidenced by $2.7 billion in spending on Business Process Management Systems (BPMS) (Gartner, 2015). The automation of a business process according to a set of procedural rules is known as a workflow (WfMC, 1999). A workflow management system (WfMS), defines, creates, and manages the execution of workflows (WfMC, 1999). BPMN (Business Process Model and Notation) (OMG, 2011), supports Business Process Modeling (BPM) with a common notation standard. 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.

In software development, one observable agility trend is the widespread application of the microservice architecture style (Fowler and Lewis, 2014) for 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 more a automated and dynamic approach to service orchestration becomes evident.

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, rather than pursue the heavyweight Service-Oriented Architecture (SOA) and semantic web, we chose a 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, 2017) using cognitive agents and investigated its resource usage and viability (Oberhauser, 2016). This paper contributes enhanced support for business modeling with Microflows and microservices, providing bi-directional support for graphical modeling with BPMN via automated constraint extraction and BPMN generation from a Microflow execution log. Furthermore, automated error handing and replanning capabilities were extended to address the dynamic microservice landscape. Note that this approach is not intended to address all facets of BPMS support, but focused on a narrow area addressing the automatic orchestration of dynamic workflows given a multitude of microservices 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

Microflow is used in IBM business process manager terminology to mean a transient non-interruptible BPEL (Web Services Business Process Execution Language) process (IBM, 2015), while 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 business process modeling 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 & 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.

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 is supported and error handling permits dynamic recovery and replanning.

# 3 SOLUTION APPROACH

The principles and process constituting the solution approach, based on (Oberhauser, 2016) and (Oberhauser, 2017), are elucidated below and reference the solution architecture of Figure 1. 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.

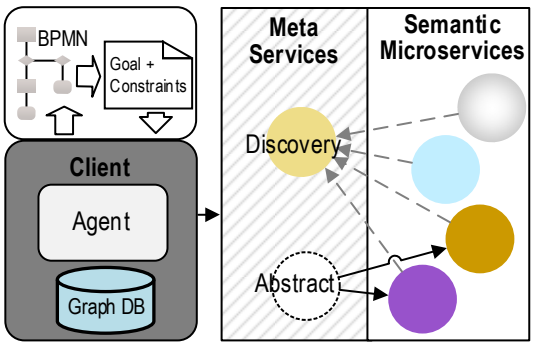


Figure 1: Solution concept.

## 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 Figure 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 Figure 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 Figure 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 Figure 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 Figure 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.

*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

The Microflow lifecycle involves five stages as shown in Figure 2.


Figure 2: Microflow lifecycle.

For the *Microflow Modeling* stage, goal and constraint specifications are modeled (currently in JSON) or extracted via tools from existing business process models such as BPMN or process mining of process (or Microflow) execution logs.

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 do 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 between the start and end timepoints of this enactment lifecycle, we chose the pragmatic and hopefully more lightweight approach from the 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

Figure 3 shows our realization of the Microflow solution concept with a mapping of primary technology choices in our prototype. 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 new extensions.

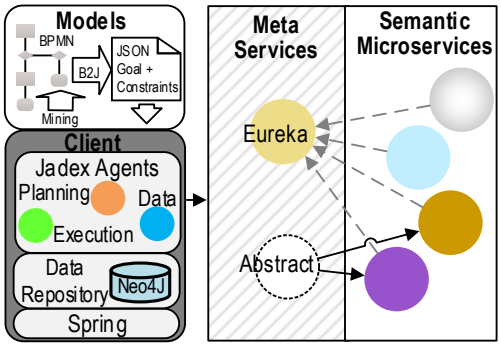


Figure 3: Microflow prototype realization.

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) was 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 (Eureka, 2016) v. 1.1.147 is used.

The microservice 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 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 Transformation

A BPMN-Microflow transformation tool (B2J in Figure 3) was implemented in Java that parses BPMN 2.0 files, automatically extracting the start and end node (goal) and any constraints, generating a Microflow JSON file. The java libraries camunda-bpmn-model and camunda-xml-model version 7.6.0 were 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).

## 4.2 Microflow Constraint Mining

A MicroflowLog-BPMN mining tool (represented by Mining in Figure 3) was implemented in Java that automatically parses our Microflow execution log file and generates a BPMN 2.0 file. 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 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 Figure 4 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.

```
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
```

Figure 4: Microflow execution algorithm.

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.

# 5 EVALUATION

A case study 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, and then error recovery.

## 5.1 BPMN Transformation

As an illustrative example, we created our own travel booking process shown in Figure 5, whereby both a hotel and flight should be found, whereafter 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 equivalent BPMN model (Figure 7) 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 (Figure 5).

```
{ "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" }]}
```

Figure 5: Travel booking example Microflow constraints.

```
{"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"}]}
```

Figure 6: SubProcess BPMN extracted constraints.

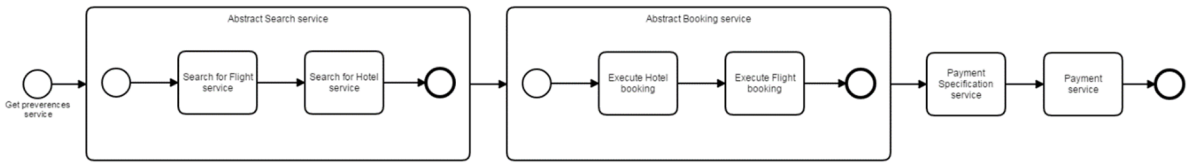


Figure 7: Travel booking example as BPMN.

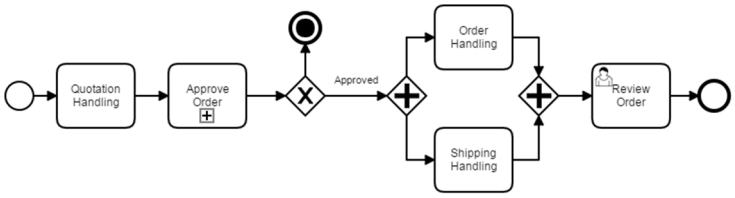


Figure 8: Collapsed SubProcess BPMN model.

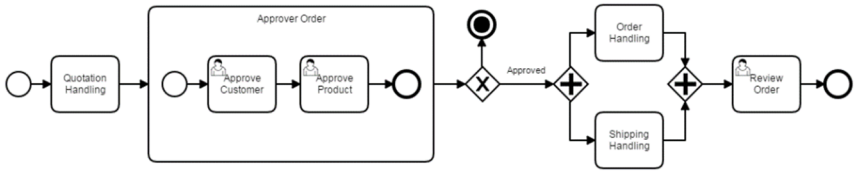


Figure 9: Expanded SubProcess BPMN model.

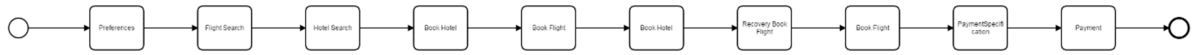


Figure 10: BPMN process mined from Microflow execution log containing a recovery case.

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 as well as the Expanded SubProcess BPMN models shown in Figure 8 and 9 respectively 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 Figure 6. Both BPMN files contain the subprocess information which is hidden in the graphical representation in Figure 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

Our MicroflowLog-BPMN mining tool was used to extract a BPMN file from our execution log (Figure 11) based on the Figure 5 and Figure 7 example that included an automated error recovery condition. Figure 10 shows the graphical BPMN representation and Figure 13 an extract from its BPMN file. As explained in Section 4.2, this can assist human analysis or serve as a starting point for a model.

```
Plan: (7)-[CAN_CALL,6]->(5)-[CAN_CALL,5]->(6)-[CAN_CALL,11]->(5)-[CAN_CALL,4]->(4)-
[CAN_CALL,2]->(5)-[CAN_CALL,3]->(1)-[CAN_CALL,8]->(8)-[CAN_CALL,10]->(1)-[CAN_CALL,9]
->(0)-[CAN_CALL,0]->(1)-[CAN_CALL,7]->(3)-[CAN_CALL,1]->(2)
Executing: http://178.18.9.151:8333/ (Get preferences service) Received: ItemList
Skipping Abstract Node:http://178.18.9.151:8335/ (Abstract Search service)
Executing: http://178.18.9.151:8340/ (Search for Flight service) Received: Flight
Skipping Abstract Node:http://178.18.9.151:8335/ (Abstract Search service)
Executing: http://178.18.9.151:8339/ (Search for Hotels service) Received: Hotel
Skipping Abstract Node:http://178.18.9.151:8335/ (Abstract Search service)
Skipping Abstract Node:http://178.18.9.151:8334 (Abstract Booking service)
Executing: http://178.18.9.151:8336/ (Execute Hotel booking) Received: LodgingReservation
Skipping Abstract Node:http://178.18.9.151:8334 (Abstract Booking service)
Executing: http://178.18.9.151:8337/ (Execute Flight booking)
ERROR: Flight
Restart with new path at index 6
Plan: (7)-[CAN_CALL,6]->(5)-[CAN_CALL,5]->(6)-[CAN_CALL,11]->(5)-[CAN_CALL,4]->(4)-
[CAN_CALL,2]->(5)-[CAN_CALL,3]->(1)-[CAN_CALL,8]->(8)-[CAN_CALL,10]->(1)-[CAN_CALL,12]
->(10)->[CAN_CALL,13]->(0)-[CAN_CALL,0]->(1)-[CAN_CALL,7]->(3)-[CAN_CALL,1]->(2)
Restore availableInput for node index: 6
Skipping Abstract Node:http://178.18.9.151:8334 (Abstract Booking service)
Executing: http://178.18.9.151:8336/ (Execute Hotel booking) Received: LodgingReservation
Skipping Abstract Node:http://178.18.9.151:8334 (Abstract Booking service)
Executing: http://178.18.9.151:8357/ (Recovery for Flight booking) Received: FlightBookingRecoveryReport
Executing: http://178.18.9.151:8337/ (Execute Flight booking) Received: FlightReservation
Skipping Abstract Node:http://178.18.9.151:8334 (Abstract Booking service)
Executing: http://178.18.9.151:8341/ (Payment Specification service) Received: PaymentChargeSpecification
Executing: http://178.18.9.151:8338/ (Payment service) Received: PaymentStatusType
```

Figure 11: Log file output (highlighting recovery in bold).


```
{ "startServiceType":"StartProcess",
"endServiceType":"EndProcess",
"constraints":[
 { "type":"BeforeNode",
  "target":"PaymentSpecification",
  "constraint":"Payment"},
 { "type":"BeforeNode",
  "target":"Book Flight",
  "constraint":"PaymentSpecification"},
 { "type":"BeforeNode",
  "target":"Recovery Book Flight",
  "constraint":"Book Flight"},
 { "type":"BeforeNode",
  "target":"Book Hotel",
  "constraint":"Recovery Book Flight"},
 { "type":"BeforeNode",
  "target":"Book Flight",
  "constraint":"Book Hotel"},
 { "type":"BeforeNode",
  "target":"Book Hotel",
  "constraint":"Book Flight"},
 { "type":"BeforeNode",
  "target":"Hotel Search",
  "constraint":"Book Hotel"},
 { "type":"BeforeNode",
  "target":"Flight Search",
  "constraint":"Hotel Search"},
 { "type":"BeforeNode",
  "target":"Preferences",
  "constraint":"Flight Search"}]}
```


Figure 12: Constraints from Travel Booking BPMN.


```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions id="Definition_1" targetNamespace="http://bpmn.io/schema/bpmn" xmlr
<bpmn:process isExecutable="false" id="_1-">
 <bpmn:startEvent name="" id="StartProcess">
  <bpmn:outgoing>_2-0</bpmn:outgoing>
 </bpmn:startEvent>
 ...
 <bpmn:task completionQuantity="1" isForCompensation="false" startQuantity="1" name=
  <bpmn:incoming>_2-1</bpmn:incoming>
  <bpmn:outgoing>_2-2</bpmn:outgoing>
 </bpmn:task>
 ...
 <bpmn:task completionQuantity="1" isForCompensation="false" startQuantity="1" name=
  <bpmn:incoming>_2-9</bpmn:incoming>
  <bpmn:outgoing>_2-10</bpmn:outgoing>
 </bpmn:task>
 <bpmn:endEvent name="" id="EndProcess">
  <bpmn:incoming>_2-10</bpmn:incoming>
 </bpmn:endEvent>
 <bpmn:sequenceFlow sourceRef="StartProcess" targetRef="_1-0" name="" id="_2-0"/>
 <bpmn:sequenceFlow sourceRef="_1-0" targetRef="_1-1" name="" id="_2-1"/>
 ...
 <bpmn:sequenceFlow sourceRef="_1-9" targetRef="EndProcess" name="" id="_2-10"/>
</bpmn:process>
<bpmndi:BPMNDiagram id="BPMNDiagram_1">
 <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="_1-">
  <bpmndi:BPMNShape id="StartEvent_StartProcess" bpmnElement="StartProcess">
   <dc:Bounds x="250" y="222" width="36" height="36"/>
  </bpmndi:BPMNShape>
  ...
  <bpmndi:BPMNShape id="Task__1-8" bpmnElement="_1-8">
   <dc:Bounds x="2000" y="200" width="100" height="80"/>
  </bpmndi:BPMNShape>
...
  <bpmndi:BPMNShape id="EndEvent_EndProcess" bpmnElement="EndProcess">
   <dc:Bounds x="2400" y="222" width="36" height="36"/>
  </bpmndi:BPMNShape>
  <bpmndi:BPMNEdge id="SequenceFlow_2-0" bpmnElement="_2-0">
   <di:waypoint xsi:type="dc:Point" x="286" y="240"/>
   <di:waypoint xsi:type="dc:Point" x="400" y="240"/>
  </bpmndi:BPMNEdge>
...
 </bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>
```


Figure 13: BPMN from Travel Booking log file with error.

Though not necessarily useful, this extracted BPMN was then converted to Microflow constraints as shown in Figure 12 to demonstrate that a full cycle back to a Microflow specification from an execution log is feasible. These constraints could, for example, then be reduced by a human to only those required and adjusted for requisite sequencing in order to utilize the dynamic planning capability.

## 5.3 Microflow Error Recovery

To demonstrate the automated error recovery capability, the Flight Booking service was modified to return an HTTP 500 status code and a Recovery for Flight Booking microservice (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.

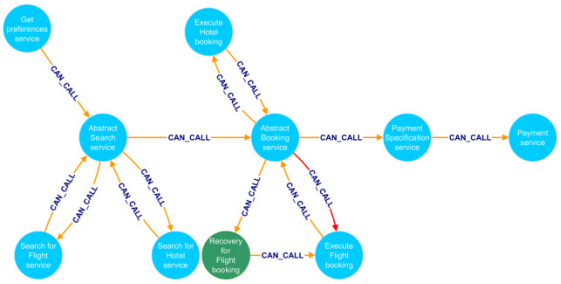


Figure 14: Travel Booking example as Neo4J graph (error recovery shown in green).

```
1  FlightReservation:
2   -
    {"reservationFor":{"departureTime":"","departureTerminal":"","departureAirport":{"iataC
    ode":"","icaoCode":"","@type":"Airport"},"departureGate":"","@type":"Flight","arrivalTi
    me":"","aircraft":"","arrivalAirport":{"iataCode":"","icaoCode":"","@type":"Airport"},"
    arrivalGate":"","arrivalTerminal":"","flightNumber":""},"bookingTime":"","boardingGroup
    ":"","reservationId":"","totalPrice":0,"@type":"FlightReservation","@context":{"@vocab"
    :"http://schema.org/"},"reservationStatus":"https://schema.org/ReservationConfirmed","u
    nderName":""}
3  LodgingReservation:
4   -
    {"reservationFor":{"address":"","@type":"Hotel","name":""},"bookingTime":"","checkinTim
    e":"","numAdults":0,"totalPrice":0,"@type":"LodgingReservation","@context":{"@vocab":"h
    ttp://schema.org/"},"reservationId":"","lodgingUnitDescription":"","numChildren":0,"che
    ckoutTime":"","reservationStatus":"https://schema.org/ReservationConfirmed","underName"
    :""}
5  Flight:
6   -
    {"departureTime":"","departureTerminal":"","departureAirport":{"iataCode":"","icaoCode"
    :"","@type":"Airport"},"departureGate":"","@type":"Flight","arrivalTime":"","aircraft":
    "","@context":{"@vocab":"http://schema.org/"},"arrivalAirport":{"iataCode":"","icaoCode
    ":"","@type":"Airport"},"arrivalGate":"","arrivalTerminal":"","flightNumber":""}
7  Hotel:
8   - {"address":"","@type":"Hotel","name":"","@context":{"@vocab":"http://schema.org/"}}
9  ItemList:
10  - {"@type":"ItemList","numberOfItems":0,"@context":{"@vocab":"http://schema.org/"}}
```

Figure 15: Output of client state in JSON.

Figure 14 includes a recovery microservice (green). In the execution log file of Figure 11, after receiving an error the execution returns to Abstract Booking Service. The client state (shown in Figure 15) is restored to that which it was at the last commit, leaving ItemList, Hotel, and Flight (Lines 5-10) and discarding LodgingReservation and FlightReservation (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 Figure 11 with the difference in the planning sequence from [CAN_CALL,9] to [CAN_CALL,12]-->(10)-->[CAN_CALL,13].

# 6 CONCLUSIONS

In this paper, we described business process modeling integration with Microflows, an automatic lightweight declarative approach for the workflow-centric 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. 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, 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 quite 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.

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, and an empirical and industrial usage.

# ACKNOWLEDGEMENTS

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

# REFERENCES


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

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

Bouguettaya, A., Sheng, Q.Z. and Daniel, F. (2014). *Web services foundations*. Springer.

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

Eureka (2016). Retrieved May 7, 2017 from: https://github.com/Netflix/eureka/wiki

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

Florio, L. (2015). 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.

Fowler, M. & Lewis, J. (2014). Microservices a definition of this new architectural term. Retrieved May 7, 2017 from: http://martinfowler.com/articles/microservices.htm

Gartner (2015). Gartner Says Spending on Business Process Management Suites to Reach $2.7 Billion in 2015 as Organizations Digitalize Processes. Press release. Retrieved May 7, 2017 from: https://www.gartner.com/newsroom/id/3064717

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

IBM (2015). IBM Business Process Manager V8.5.6 documentation. Retrieved May 7, 2017 from: http://www.ibm.com/support/knowledgecenter/SSFPJS_8.5.6/com.ibm.wbpm.wid.bpel.doc/topics/cprocess_transaction_micro.html

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

Lanthaler, M. (2013). Creating 3rd generation web APIs with hydra. In *Proceedings of the 22nd international conference on World Wide Web companion*. International World Wide Web Conferences Steering Committee, pp. 35-38.

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

Lanthaler, M. and Gütl, C. (2013). 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.

Martin, D. et al. (2004). *OWL-S: Semantic markup for web services*. W3C member submission, 22, pp.2007-04.

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

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

Oberhauser, R. (2016). 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.

Oberhauser, R. (2017). 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.

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

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

Rajasekar, A., Wan, M., Moore, R., & Schroeder, W. (2012). 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.

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

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

Singer, R. (2016). 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* (p. 8). ACM.

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

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

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.