2017 43rd Euromicro Conference on Software Engineering and Advanced Applications

Actor Based Business Process Modeling and Execution: a Reference Implementation Based on Ontology Models and Microservices

Matthias Geisriegler*, Maksym Kolodiy*, Stefan Stani*, and Robert Singer†
FH JOANNEUM – University of Applied Sciences
Dep. of Applied Computer Sciences
Alte Poststraße 147
8020 Graz, Austria
robert.singer@fh-joanneum.at

***Abstract*—In this work, we present a proposal for a reference architecture for the modeling and execution of business processes. We define business processes following the subject-oriented business process management (S-BPM) methodology, which recently has been amalgamated in a Web Ontology Language (OWL) definition. Consequently, all created process definitions are serialized as OWL files. The S-BPM approach understands business processes as a network of distributed and independent actors (human or machine) which interact via the exchange of messages. The proposed architecture is therefore designed as a collection of microservices. All concepts have been realized as a prototypical software application (modeling and execution of business processes) for reference and further research. It is proved that the generated OWL business process models can be executed on the developed workflow engine.**

I. Introduction

Current developments in business, the demand for digitalization, and technology driven new business models foster more than ever the need for mature business process management (BPM) methodologies and corresponding supporting technologies for distributed business processes, so-called choreographies. Communication is the very nature of a business process choreography—or, in other words, any choreography is a set of structured communication patterns. That means a choreography defines how work is done, taking into account all involved process participants and systems.

During recent years, tools have emerged to support the execution of business processes, so-called Business Process Management Suits (BPMS). Most of these tools are built around the *de facto* standard for business process modeling languages, namely Business Process Model and Notation (BPMN 2.x).

The standard may be suitable for modeling purposes, but does not directly support the execution of business process models [1] [2] [3]. That means, there is a gap between the conceptual model and the digitized and executable representation. For example, the BPMN standard document provides several compliance classes. Though, the class for execution offers only a rather restricted subset of the modeling classes and no execution support for process collaborations and choreographies [1].

To overcome this weakness, modeling notations based on actor models have emerged. The standard for an actor based approach for business process modeling is the so-called Subject-oriented Business Process Management (S-BPM) approach [4] which not only provides a formal but rather simple, modeling notation and also allows a direct translation into executable code. S-BPM is a mature approach, as has been proven in theory and practice [5] [4] [6] [7].

In this paper, we will discuss how to move away from monolithic software suites towards agile approaches. The central research question is, how to define a BPMS architecture which is fully based on microservice design principles.

To have a common understanding, we will shortly discuss some topics for a shared understanding of the problem domain. Then we will discuss an architecture based on microservice for the execution of process models; the discussion builds on a prototypical implementation of the presented architecture, which is also available for further research. We will not argue why in this case a software architecture based on microservices is more adequate than a monolithic one; we take this for granted and as state of the art for enterprise applications [8] [9].

II. Workflow Systems

A typical conceptual architecture of an enterprise workflow system as part of a business process management system is based on a monolithic architecture, as described in [10], for example. All known workflow-engines are based on these architectural concepts; therefore, most of them are heavy-weight applications with steep learning curves. Moreover, many of these applications have ”problems” to realize process collaborations in a practical way. This can easily be proved by trial and is rooted in the underlying concept to conceptualize a business process as *one* finite state machine (FSM).

*Equal contribution
†Corresponding author

978-1-5386-2141-7/17 $31.00 © 2017 IEEE
DOI 10.1109/SEAA.2017.39

III. Subject-oriented Business Process Management (S-BPM)

A different approach has been proposed by Fleischmann [5] who developed a business process modeling methodology called Parallel Activity Specification Scheme (PASS), which itself is based on the Calculus of Communicating Systems (CCS) [11] [12], a process calculus describing reactive systems [13]. Later, this concept has evolved into the so-called Subject-oriented Business Process Management (S-BPM) methodology [4]. In short, this methodology includes all necessary concepts to define reactive and executable models of business processes.

An S-BPM process is defined via the communication exchange between subjects (actors are instantiated subjects in this context, or the other way round—subjects are generalizations of actors). Additionally, each subject has a defined (but invisible to the outside world) internal behavior, which is determined as a process flow using states for receiving or sending a message (to another subject), and states in which the subject is doing some work. States can be flagged as starting or ending states and are connected using directed arcs. In our approach, we think of subjects as actors. In the context of BPM, actors define who is doing what, as they are mapped to a resource for execution (organizational roles). Typically, S-BPM models consist of two types of representations: a Subject Interaction Diagram (SID) and a set of Subject Behavior Diagrams (SBD). The SID includes the subjects (this are the actors), the messages exchanged between the actors and the business objects attached to the messages. The SBD includes all possible state sequences of an actor: a finite set of send, receive and function states.

According to our definition, any actor can be represented as a Finite State Machine (FSM) with states as mentioned above, the so-called internal behavior. Furthermore, state changes can be triggered by receiving messages from other actors [14]. A subject is a general concept and can be instantiated by a human or machine.

The semantics of S-BPM is amalgamed in an OWL[1] ontology, so that process models can be shared between workflow engines as long as the models are based on the standard ontology. The S-BPM ontology[2] and its conceptual background is discussed in [15].

IV. S-BPM Execution Platform

A. Previous Work

Previously we have presented an entirely functional S-BPM workflow solution[3] using the Microsoft Windows Workflow Foundation functionality as discussed in [16] and [17]. The designed architecture has full enterprise functionality and supports the execution of inter- and intra-company business processes (i.e. collaborations, choreographies).

[1]https://www.w3.org
[2]https://www.i2pm.net
[3]https://github.com/InFlowBPM/InFlow-BPMS

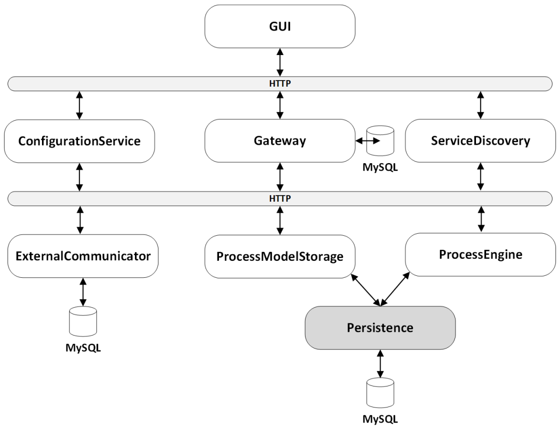


Fig. 1. The architecture of the prototype: a collection of microservices.

Nevertheless, the developed prototype is still based on the concepts of traditional workflow systems—e.g. *one* web server for all process instances— and needs many infrastructure components as a functional backbone and is therefore rather costly and has a rather high complexity.

B. Prototype based on Microservices

One major design goal of our architecture is to separate the modeling and the execution platform. As a result, the modeling platform generates an OWL file that can be uploaded to the execution platform by using the management user interface (UI). The management UI provides an interface to control the execution of processes. The individual platforms are based on different sets of technology, as will be described next.

This chapter includes basic information about the execution platform. Firstly, the utilized technologies for the creation of the various services within the execution platform are shown, and secondly, the single services are described in more detail.

The core of the execution platform (based on Java) is built on the following technologies:

- Spring Boot/Cloud is the dominant technology in the execution platform. This technology enables easy creation of standalone applications based on the Spring technology stack. Furthermore, it includes an embedded webserver, e.g. Tomcat in this case.
  Due to Spring Boot, the creation of separate services is simplified. The data interfaces between the services are based on Representational State Transfer (REST) and JavaScript Object Notation (JSON). Therefore, a platform independent communication between different services and external systems is possible.
- For the storage of the process data, a MySQL database is used.
- The mapping of the database tables and the Java objects is done by using Hibernate. This object/relational mapping (ORM) framework is using metadata that describes the mapping between the classes of the application and the schema of the SQL database [18].

The basic architecture of the execution platform is visualized in Figure 1. As can be seen, the architecture depends on the following services:

- Graphical User interface (GUI)
- Configuration Service
- Gateway
- Service Discovery
- External Communicator
- Process Model Storage
- Process Engine

1) Graphical User interface (GUI): The main features of the GUI are: user authentication, starting and executing processes, visualization of KPIs, and import of process models via OWL files

The UI platform communicates and interacts with the Gateway via REST calls. This loose coupling allows an easy replacement of components, as long as the REST interfaces remain unchanged. On the one hand, the Process Engine or the Process Model Repository could be replaced without changing the UI platform. On the other hand, the UI could be replaced easily as well.

2) Configuration Service: Microservice based architectures need a central repository—in our case a GIT-repository—for all configuration files. The service is based on Spring Cloud Config which supports the management of configuration files in distributed systems.

3) Gateway: As has already been mentioned above this service acts as a router to forward requests or responses to the correct recipient. Additionally, the Gateway provides an authentication service to use external authentication providers like Active Directory. Users can have one or more roles, and each role can be assigned to one or more rules (permissions). Our authentication principle follows an extended role based access control model (RBAC with rule groups).

Users can log in on the UI platform and authenticate themselves via so-called JSON Web Tokens[4] (JWT). JWT is an open standard for securely transmitting information between parties as a JSON object. After providing the credentials to a REST interface, the user gets a JWT that will be included in every subsequent request. This token will be saved in the local storage of the browser, which enables a stateless authentication mechanism.

4) Service Discovery: In a typical microservice ecosystem, a service registration & discovery is necessary which is provided by the ServiceDiscovery. Each service registers itself with the ServiceDiscovery and tells the registry where it lives (host, port, node name). The clients send requests to the ServiceDiscovery to retrieve information about the other services.

5) ExternalCommunicator: The primary goal of the ExternalCommunicator is to add support for external subjects. Thus, it enables to send requests to external systems and receive responses from external systems. The External Communicator is based on parser implementations for incoming requests and composer implementations for outgoing requests. At the moment implementations for JSON and XML are available.

[4]https://jwt.io

6) ProcessModelStorage: An OWL file, which describes an S-BPM process model according to the semantic specifications of the standard-pass-ont [15], can be imported. This OWL file is parsed via Apache Jena, an open source framework for semantic web in Java.

7) ProcessEngine: The ProcessEngine is the most extensive service of the execution platform. This service provides features to start, stop processes, and handles the complete workflow of processes. The Process Engine is based on Akka, which is one of the most popular Actor Model frameworks.

In the Actor Model, all objects are independent, computational units. These units only respond to received messages and do not share a common state. Actors change their state only when they receive a stimulus in the form of a message. So, an actor is a computational entity that, in response to a message it receives, can concurrently [19]:

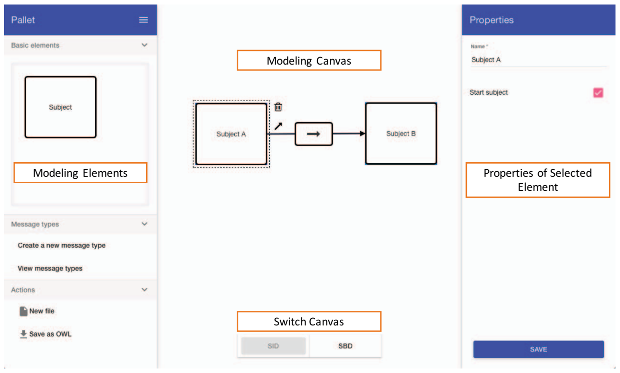
- send a finite number of messages to other actors
- create a finite number of new actors
- designate the behavior to be used for the next message it receives

The Akka framework is based on the Actor Model concept. In general, it is an event-driven, middleware framework to build concurrent, scalable, distributed applications. Concurrent programs are split into separate entities that work on distinct subtasks. Actors can change their state and behavior based on the message passed. This allows them to respond to changes in the received messages [19]. In S-BPM subjects change its state depending on the received message. So basically, the concepts of Actor Models and S-BPM are quite similar.

The actor system can be split into two major parts: (1) The User Supervisor Actor that contains any number of user actors and (2) the Process Supervisor Actor which consists of process actors. This approach implements the Akka supervisor strategy and provides a clear separation of duties between process and user responsibilities. Furthermore, tasks (also an actor) can be executed by supervisor-, user and process actors to outsource complex activities to an independent actor to ensure that other actors are not affected by other tasks. Lastly, the Analyze Actor provides different key performance indicators (KPIs), e.g. some processes finished in a specified time frame.

The responsibilities of each actor are defined as follows:

- Process Supervisor Actor: It is responsible for starting and stopping processes.
- Process Actor: This actor handles all messages to retrieve the current state (e.g. running, finished, canceled) of a process. It provides functions to check whether a process is still active.
- User Supervisor Actor: Initializes new actors for each user based on its user identifier. So, for each user, one user actor is created. This actor also handles wake up messages and acts as a router for the underlying user actors.
- User Actor: It returns the open tasks of one specific user. Additionally, the actor provides state objects for a user in



Name *
Subject A
Start subject ✓

Fig. 2. SID view of the modeling platform.

a running process. A state object contains all information what can be done by a user. This information is process specific. So, a state object consists of business objects and returns the next possible states. Furthermore, it handles state object change messages, where the state of one user is changed. The actor also handles retrieved messages from other users.

- Tasks: Task provides a way to outsource complex activities like changing the subject state or storing business objects. User Supervisor Actor, Process Supervisor Actor, User Actor, Process Actor can execute tasks. Tasks can be invoked by the usage of the task manager.
- Analyze: Actor which executes SQL queries to retrieve KPIs of the Process Engine.

V. Modeling Platform

The modeling platform (see Figure 2) was created with the goal to provide an Open-Source platform that can be easily used to model S-BPM processes and export them as an OWL file. The platform is built as a web-based application with the intention to provide a platform that is independent of any operating system. Consequently, a stack of different technologies was selected to enable the implementation of illustrated requirements. The modeling platform can be easily maintained and should be understood as a playing ground for research in the domain of business process modeling based on OWL definitions.

Alternatively, there are shapes[5] for Microsoft Visio available to define S-BPM models which can be exported as OWL files and directly loaded into the workflow prototype for execution.

VI. Discussion

With this prototype, we want to stimulate further research in the domain of business process modeling and execution (the prototype is released as open source software). The presented and discussed architecture (which is still under development at the time of writing) is a starting point to define a modern reference architecture for BPMS which are based on microservices.

[5]https://subjective-me.jimdo.com/downloads/

The microservice approach supports several real requirements for enterprise software: with this work, we have a proof of concept, which demonstrates the practical usability of the microservice approach in the domain of business process management, in concrete in the development of agile workflow systems. Nevertheless, the pre-condition to building an agile BPMS is to change the way how to define process models.

References

[1] Object Management Group, "Business Process Model and Notation," http://omg.org, 2013.
[2] B. Silver, *BPMN Method and Style*, 2nd ed. Cody-Cassidy Press, 2011.
[3] E. Börger, "Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL," *Software & Systems Modeling*, 2011.
[4] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger, *Subject-Oriented Business Process Management*. Springer, 2012.
[5] A. Fleischmann, *Distributed Systems: Software design and implementation*. Springer, 1994.
[6] A. Fleischmann, S. Raß, and R. Singer, *S-BPM Illustrated*. Springer, 2013.
[7] A. Fleischmann, W. Schmidt, and C. Stary, Eds., *S-BPM in the Wild*. Springer, 2015.
[8] M. Amundsen and M. Mclarity, *Microservice Architecture*. O'Reilly, 2016.
[9] S. Newman, *Building Microservices*. O'Reilly, 2015.
[10] D. Hollingsworth, *The Workflow Reference Model*. Workflow Management Coalition, 1994.
[11] R. Milner, *A Calculus of Communicating Systems*, ser. LNCS. Springer, 1980, vol. 92.
[12] S. Borgert, J. Steinmetz, and M. Mühlhäuser, "ePASS-IoS 1.1: Enabling Inter-enterprise Business Process Modeling by S-BPM and the Internet of Service Concept," in *S-BPM ONE: Learning by Doing - Doing by Learning*, ser. CCIS, W. Schmidt, Ed., vol. 213. Springer, 2011, pp. 190–211.
[13] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba, *Reactive Systems*. Cambridge Univ. Press, 2007.
[14] D. Brand and P. Zafiropulo, "On Communicating Finite-state Machines," *Journal of the Association for Computer Machinery*, vol. 30, no. 2, pp. 323–342, 1983.
[15] M. Elstermann, "Proposal for Using Semantic Technologies as a Means to Store and Exchange Subject-oriented Process Models," in *Proceedings of the 9th International Conference on Subject-Oriented Business Process Management*, ser. S-BPM ONE '17. ACM, 2017.
[16] R. Singer, J. Kotremba, and S. Rass, "Modeling and Execution of Multienterprise Business Processes," in *IEEE 16th Conference on Business Informatics (CBI)*, vol. 2, 2014, pp. 68–73.
[17] R. Singer and S. Raß, "Structured Communication—Approaching S-BPM with Microsoft Technologies," in *S-BPM in the Wild*, A. Fleischmann, W. Schmidt, and C. Stary, Eds. Springer, 2015, pp. 235–255.
[18] C. Bauer, G. King, and G. Gregory, *Java Persistence with Hibernate*. Manning, 2016.
[19] M. K. Gupta, *Akka Essentials*. Packt Publishing, 2012.