A Survey on Service Composition Languages

Mahdi Saeedi Nikoo
Eindhoven University of Technology
Eindhoven, The Netherlands
m.saeedi.nikoo@tue.nl

Önder Babur
Eindhoven University of Technology
Eindhoven, The Netherlands
o.babur@tue.nl

Mark van den Brand
Eindhoven University of Technology
Eindhoven, The Netherlands
m.g.j.v.d.brand@tue.nl

ABSTRACT

In recent years, service-oriented architecture (SOA) has been adopted by industry in developing enterprise systems. Web service composition has been one of the challenging topics in SOA. Numerous approaches have been proposed to tackle this problem. In industry big companies such as Amazon, Netflix, and Uber have developed their own web service composition languages and tools. In academia, on the other hand, there have also been attempts to resolve some of the complexities in web service composition. In this survey we identify and evaluate current prominent service composition languages, and discuss our key findings. After a scan of dozens of service composition systems, 14 systems that used a language-based approach were included in this study. We believe that our findings will help people from industry and academia to learn about some of the major active composition languages and get an overall idea about their commonalities and differences.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**; **Orchestration languages**.

KEYWORDS

Service Oriented Architecture, service composition, orchestration, choreography, domain-specific language

ACM Reference Format:
Mahdi Saeedi Nikoo, Önder Babur, and Mark van den Brand. 2020. A Survey on Service Composition Languages. In *MLE 2020: International Workshop on Modeling Language Engineering and Execution, Oct 18 – 23 2020, Montreal, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Service-Oriented Architecture (SOA) is a style of software design where business solutions are built based on a network of modular components with each component implementing a specific functionality [24]. These components are called services which can be distributed across enterprises and different information systems. Web services have been growing in popularity in enterprise distributed computing, becoming a major concept in software engineering. These services are built based on open standards and are loosely-coupled, which allows them to be reconfigured to address new business requirements [22].

One important aspect of SOA is to see how these services are composed to work together. The aim of service composition is to build a composite service that consists of multiple loosely-coupled web services to address more complex business requirements [17]. The way these services are composed to build the composite service is generally described by a workflow.

The goal of this paper is to provide a high-level analysis of the current service composition languages. As an initial step towards a more comprehensive evaluative and analytical study, this work should provide the readers with some insights in their evaluation of current service composition systems and languages and help them decide better on what language to choose. We believe that our findings will help people from industry and academia to learn about some of the major active composition languages and get an overall idea about their commonalities and differences. We aim to answer the following research questions:

RQ1 What are the available service composition languages?
RQ2 Which ones are being actively used?
RQ3 What are their major characteristics with respect to SOA and language engineering aspects?

The remainder of the paper is structured as follows. Section 2 gives an explanation of the main service composition approaches and techniques. Section 3 discusses the related work. In Section 4, the available composition languages are presented. Section 5 presents our analysis and evaluation of these languages and the results obtained. Section 6 mainly provides a discussion based on the obtained results and gives some suggestions for the future directions. Finally Section 7 concludes the study.

2 SERVICE COMPOSITION APPROACHES

Web service composition is the process of aggregating multiple services over the web to provide more complex functionality. In this section, we briefly explain some of the key terminology and aspects of service composition related to our study.

Web Services The term web service is commonly used nowadays to refer to mainstream SOA applications. Although there are different definitions for the term, it is usually seen as an application accessible on the web. IBM defines web services as "a new breed of Web application, and they are self-contained, self-describing, modular applications that can be published, located and invoked across the Web".

Composition via Orchestration vs. Choreography Composition can involve two major models: orchestration and choreography. Service orchestration looks at the business process from a single central point of view and provides an executable flow that

MLE 2020, Oct 18 – 23 2020, Montreal, Canada

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

coordinates the interactions among services. It therefore is a mediated approach in which all the service dependencies are known to the orchestrator. The orchestrator takes care of invoking and combining the services. The relationship between all the services participating in the process are described by a single endpoint, i.e the composite service.

Service choreography is of more collaborative nature, and looks at the business process from a global point of view. It gives a global description of the observable behaviour of each of the participating services in the process which is defined through message exchange, rule of interaction and agreements between two or more endpoints. It allows each of the involved parties to specify their part in the interaction. Choreography applies a decentralized and collaborative approach for service composition.

Static vs. Dynamic Composition Based on the time of service composition, the composition can be either static or dynamic. In static approach, the service aggregation takes place at design time. This is suitable for the case where the partners involved in the process are relatively fixed and service functionalities or composition requirements do not change or rarely change. In contrast, in dynamic composition, depending on the composition requirements, the service components are allowed to change at runtime. This type of composition is increasingly preferred over static type, as it allows for handling unpredictable changes at runtime.

Manual vs. (Semi) Automated Composition This aspect deals with whether the composition is done manually by a designer or automated using a composition system. Because of the behaviour dynamism and the flexibility of the web, manual composition can be complex and error-prone, and does not guarantee the satisfaction of user requirements at runtime. In contrast, automated composition approaches consider user requests and automatically do the composition to achieve this goal [5]. Automated approaches typically leverage semantic web and artificial intelligence (AI) planning techniques. Realizing the fully automated service composition is difficult in practice and there are several issues with them [6].

3 RELATED WORK

There are quite a few survey papers on service composition in general (without explicit focus on the languages) over the years; we mention mostly the recent ones here. Huf et al. [14] present a wide survey on service composition approaches with a focus on service type heterogeneity and the applied methods. Hamzei et al. in [13] look at service composition techniques in IoT which they categorize into four classes, namely framework, SOA and RESTful, heuristic, and model-based. Lemos et al. in [16] survey the state-of-the-art from the aspect of techniques and tools in web service composition. They provide an analysis framework to cover fundamental building blocks in terms of concepts, models, languages, techniques, and tools. Another work is the article by Sheng et al. in [25] that overviews the life cycle of web services composition based on a set of assessment criteria and surveys the main standards, research prototypes, and platforms in the domain. One of the few works on service composition languages was done by Bucchiarone et al. [8] which is an old one.

There are several survey papers with a few recent ones that discuss service composition systems and languages. However, recent works do not mainly consider active languages used in academia and industry. We tried to address this in our initial attempt. Also, the other aspect of our work is to give more focus on the language aspects and their evaluation from three different angles. These include general features, SOA concepts, and software language engineering (SLE) concepts [15].

4 CURRENT SERVICE COMPOSITION LANGUAGES

In this section we provide a list and brief description of current service composition languages. The selected languages are mainly coming from the academia or industry, with a few proposed by standards consortia. This section addresses RQ1 and RQ2.

In our search the main focus area was on systems and languages offering service composition. For academic works, the list of languages were mainly extracted from individual works with a language proposition and the survey papers on the topic conducted within last five years. A snowballing was also performed on the major works from the result set to learn about other potentially useful service composition platforms with a language-oriented approach. Besides the mentioned industry works in the literature, the search process was accompanied with Google search to expand the list. From the total of 38 languages identified, 14 of them were included in this survey.

For this study we tried to cover the languages that meet our inclusion criteria and are known in the domain. For the inclusion of a language two main criteria were considered. One was if the composition system is active, meaning that it is currently being used and supported by industry, open source community, or research groups in academia. This means that the list was shortened to exclude the deprecated languages, even if they were active in the past. The other criterion was to include platforms providing their own domain specific language (DSL) for service composition instead of relying on a general purpose language [29]. Besides these, two other platforms that depend on the BPMN (Business Process Model and Notation) standard were also included.

There are works such as ING Baker [1] that in a way provide an extension to an existing general-purpose language such as Java or Python to provide the means of service composition. These were omitted from the study. Similarly, other works that build on general-purpose modeling languages such as UML were not considered either. This work also does not include other types of composition such as cloud orchestration and mashups.

BPEL (Business Process Execution Language) is a language for defining business processes that allows web services to interconnect and share data [20]. Standardized by OASIS in 2004, BPEL is a convergence of two former workflow languages, Web Services Flow Language (WSFL) by IBM and XLANG by Microsoft.

BPMN 2.0 Business Process Model and Notation (BPMN) is a graphical notation and standard for business process modeling [21]. It is maintained by the Object Management Group (OMG). Version 2.0 of BPMN was released in January 2011, which introduced execution semantics to the specification. The notation of BPMN has been designed to be used by different business stakeholders including technical developers, business managers, business analysts.

[1] https://ing-bank.github.io/baker/

In this study, we also include two service composition systems, Camunda BPM, and Zeebe. These workflow engines rely on BPMN 2.0 as their workflow language. For that reason, they are not explained separately. Camunda is a platform with workflow and decision modeling with deployment and execution of workflow tasks [10]. It offers a business process management (BPM) framework that can be used as a standalone process engine server or as embedded inside Java applications. The BPMN workflow engine can be used for microservice orchestration and human task management. Zeebe is a workflow engine for microservices orchestration [31]. It allows monitoring and workflow data analysis. It also provides a language-agnostic client model, so that microservices can be built in about any programming language. Both Camunda and Zeebe provide a graphical modeling of workflows.

YAWL (Yet Another Workflow Language) is a workflow language and a software system that includes an execution engine, a graphical editor and a worklist handler [30]. It handles complex data ransformations and web service integrations. YAWL is a state-based language and is based on Petri nets to provide a basis for formal analysis of services.

Apache Taverna is a scientific workflow platform that is used for integrating many different software components with a support for SOAP- and REST-based web services [4]. It was initiated by multiple academic and industry institutions. The platform is used in several domains including bioinformatics, medicine, astronomy, and social sciences. Taverna workflows are primarily data-driven rather than control-driven and can be executed in parallel.

Jolie (Java Orchestration Language Interpreter Engine) is a programming language for developing distributed applications based on microservices [1]. It offers a C-like syntax that makes it more programmer-friendly compared to XML-based languages. Jolie allows programs defined as services to communicate over network independent from the protocols and communication medium used, or locally using in-memory communication. Jolie is based on formal orchestration process calculus.

Ballerina is a platform with a programming language for distributed programming [28]. The platform supports distributed transactions, reliable messaging, stream processing, workflows and container management. It offers a strongly-typed, concurrent environment for programming distributed systems using both textual as well as a graphical syntax based on sequence diagram metaphor.

Netflix Conductor is a cloud-based workflow orchestration engine for microservices developed by Netflix [19]. Workflows are defined in a JSON-based language that is based on tasks that are executed as part of the workflow. Tasks are either control tasks (fork, conditional etc.) or application tasks (e.g. encode a file) that are executed on a remote machine.

AWS Step Functions is based on state machines and is used to orchestrate Amazon Web Services [3]. The process is based on steps where the output of one step becomes the input of another step. The workflow which is defined using the Amazon States Language constitutes a map of all possible steps and the transitions between them. The States language is JSON-based and is proprietary to Amazon.

AIOCJ is a framework for programming distributed adaptive applications [23]. It uses Adaptive Interaction-Oriented Choreographies (AIOC), a choreographic language used for modeling the global interaction among all entities (roles) in a system. It also offers an adaptation middleware that supports adaptation of AIOC programs. It uses a rule-based approach allowing developers to specify parts of the interaction that can change.

Chor is a choreography language for developing concurrent systems based on communications [18]. It is still prototypical but has an Eclipse-based IDE support with a type checker. Programs written in Chor can be projected to a set of separate programs (in Jolie) that can be run in a distributed system.

WS-CDL (Web Services Choreography Description Language) is a specification by the W3C defined as an XML-based language for describing interactions between web services in a peer to peer manner [27]. The common and complementary behavior of the participants which is represented as ordered message exchanges among them result in accomplishing a common business goal.

Reo is a coordination language with an emphasis on connectors and their composition [11]. Reo supports loose coupling, distribution, mobility, exogenous coordination, and dynamic reconfiguration. It has a strong formal basis that allows for model checking and verification as well as strong execution semantics of a web service composition.

5 EVALUATION AND RESULTS

In the previous section, the current service composition languages that we found during this study were listed. In this section we address RQ3 and define the criteria for our evaluation. As previously mentioned, we mostly focus on the languages for our evaluation. The language provides a formalism to specify the logic of the composition which is executed as a composite application afterwards.

The evaluation consists of three dimensions that apply to the composition system: criteria that relate to general features, criteria about SOA concepts, and criteria about SLE concepts. The selected criteria for this study are mainly derived from other surveys and taxonomies such as [16], and software engineering body of knowledge [7, 9]. An explanation for each of the criteria is given below. Because of the space limitations, the full evaluation table is provided online [2].

General features: Below are the general features related to composition systems and the language they use with a short description for each.

- **Origin**: this defines where the language development was initiated which may be either standard, academic or industry origin. Out of the selected languages, there were 3 standard and 5 academic languages. The rest were from the industry.
- **Integrated Development Environment (IDE)**: This can be a stand-alone editor or a plug-in in an existing editor. Except for one of the languages (Netflix conductor), all provide a means for developing the composition. For the standard languages, there are already vendors and developers that implement tooling compliant to those standards.
- **Runtime Environment**: A key aspect in the service composition is how the composite application is executed. This feature looks at the available options such as an execution engine that is accompanied by the composition language.

[2] https://surfdrive.surf.nl/files/index.php/s/FLfn1yBqrEjDvON

For most of the languages, there is a type of runtime environment to execute the composition logic. Choreography languages, AIOCJ and Chor, do have a compilation engine instead which generate programs in Jolie language for execution.

SOA concepts: Below are the SOA concepts related to the composition systems and their language with a short description for each.

- **Composition Model:** Depending on the adopted approach, a language may be used for orchestration, choreography, or a combination of them. There are also coordination languages that can be used for these purposes.
- **Target Application:** A business process consists of activities and relationships which describe a function that accomplishes a business goal. Scientific Workflows allows scientists to define, reuse, execute, and monitor experiments and analysis of data through composition activities. Except Apache Taverna, most of the selected languages which are already popular do not explicitly aim at scientific applications, though there are works such as [2] that refer to BPEL as a suitable language for scientific compositions.
- **Dynamic Reconfiguration**: It checks if the system allows for a change or adaptation at runtime such as applying modification in a service or replacing a failed web service. With no such dynamism, the service topology needs to be specified statically. Only three composition systems, namely YAWL, AIOCJ, and Reo explicitly support this feature.

SLE concepts: This section lists the SLE concepts related to composition systems and the used language with a brief description for each.

- **Concrete Syntax:** It specifies the representation of the abstract syntax or the concrete syntax of the language. It can be textual or have graphical notations or a combination of both. Some of the languages such as Conductor, provide a DSL with a textual syntax for development, but also provide a runtime visualization to make it easier to follow the process. Many of the selected systems provide a visual syntax for designers.
- **Semantics Definition:** It refers to the language semantics and whether it is defined based on a formal theory or specified informally. A language with a formal basis paves the way for program verification and further analysis. Five of the selected languages (academic ones) explicitly offer a formal definition of language semantics, e.g. YAWL is using labelled transition system, or Jolie is based on SOCK calculus [12].
- **Formal Verification:** It shows whether the composition system allows for any formal verification for composition correctness. Verification is concerned with determining, at design time, whether a process model shows certain desirable behaviours. Only three of the selected languages, namely, YAWL, Chor, and Reo, explicitly provide a formal verification mechanism.
- **Execution Mode:** It is classified either as a compilation or interpretation. In the compilation, the source language is translated into an intermediate or target language or model of usually a lower level abstraction. In an interpretation the program is directly executed. Four of the systems follow a compilation mechanism. For the standard languages e.g. BPMN, this feature depends on how one implements a language processor.
- **Crosscutting Concerns:** Systems can support for nonfunctional features such as exception handling in the language or the encapsulating platform. Seemingly, all the systems and languages except for Chor, support exception handling for at least some extent.

6 DISCUSSION

In this section we address RQ3 and report the main insights gained as a result of our analysis:

- There has been numerous languages for service composition developed over the past years. A majority of them are either discontinued or did not grow out of their infancy.
- Many microservice architectures rely on a choreography pattern which is also known as publish-subscribe model. Despite the features such as flexibility and independence offered by choreography, there seems to be issues related to it such as visibility, flow monitoring, and error handling. It turns out that the industry still has more inclination towards using orchestration languages or a hybrid approach.
- Seemingly, because of the higher demand in industry, the number of service composition systems targeting business domain is higher compared to other domains such as scientific applications. This is also related to the fact that scientific workflows also have different requirements such as mass calculations, security, performance, and reliance on data-flow oriented mechanisms, which needs to be supported by the composition systems.
- Only three of the languages, namely YAWL, AIOCJ, and Reo, propose runtime adaptability, which is an increasingly demanding aspect for service composition systems. It also seems at the industry side, still mostly they rely on manual composition approaches which only relies on the syntactic description of services with little or no adaptation at runtime.
- Offering smart and powerful editors is always an important factor for the adoption of a software language. From our list, the industry composition systems, including those based on standards such as BPMN, are all offering a visual or a hybrid development feature. Only the very mature academic works such as YAWL provide a good visual IDE. A translation from the solutions with weaker toolset to the ones with more powerful toolset would possibly bridge this gap. That could be done e.g. through a code generation, or integration into their IDE.
- It was found out that just a few languages have a formal foundation which are basically academic works, though not all of these support a type of verification for correctness checking. It can also be said that formality and verification doesn't seem to be of deep consideration in industry solutions.
- It can be seen that all of the choreography languages are compiler-based in which projections are provided for choreography participants. This is seemingly related to how choreography systems typically work where each participant needs

to play their own local role in order for the global interaction logic to be fulfilled.
- Almost all of the approaches support crosscutting concerns such as exception handling. A few of the systems explicitly offer good scalability. Some systems such as Camunda and AIOCJ guarantee deadlock-free composition by construction.

Extension and use of this survey. Although this initial attempt provides some insights, the aim is to expand this study to a more comprehensive set of languages and features. A through study would help practitioners choose the right language for their purpose, and considering the shortcomings in the current languages and building on their strong sides, suggest possible future directions to researches, e.g. considering the heterogeneity of the services and their types, provide better interoperability mechanisms.

There are various directions for future work. Since the current study does a high-level feature comparison of the service composition systems and languages, the next step would be to look deeper into the workflow languages for service composition and do a multi-dimensional language analysis (e.g. feature analysis and clone detection on the metamodels [32]). Workflow patterns [26] can for instance provide the basic set of criteria such as control-flow, data, resource patterns for in-depth analysis. Also, from an SLE perspective, it would be useful to look at model composition appraoches which are applicable to services.

Threats to validity. Considering the validity of this survey, there are two major points. One is the fact that the current survey does not follow a systematic review methodology and thus the language result set may be biased towards the search process applied. The second is that the feature support for the systems and languages were obtained mainly based on the original publications and the corresponding online resources for them. If there's a change not explicitly reflected in these resources, that might not be covered in this survey.

7 CONCLUSION

This paper provides a high level assessment on the service composition systems with a language-oriented approach. To that end, a broad search in the domain resulted in the selection of 14 service composition systems. The result of our analysis in this paper can help readers have an overall idea about some of the major service composition systems in the literature and industry and be able to compare them based on their commonalities and differences.

ACKNOWLEDGEMENTS

This research is funded by ECSEL, the Electronic Components and Systems for European Leadership Joint Undertaking under grant agreement No 826452 (Arrowhead Tools project), supported by the European Union Horizon 2020 research and innovation programme and by the member states.

REFERENCES

[1] Patricia S. Abril and Robert Plant. 2007. JOLIE: a Java Orchestration Language Interpreter Engine. *Electronic Notes in Theoretical Computer Science* 181 (June 2007), 19–33. https://doi.org/10.1016/j.entcs.2007.01.051
[2] Asif Akram, David Meredith, and Rob Allan. 2006. Evaluation of BPEL to scientific workflows. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, Vol. 1. IEEE, 269–274.
[3] Inc. Amazon Web Services. 2020. *AWS Step Functions.* Retrieved July 20, 2020 from https://aws.amazon.com/step-functions/
[4] Apache. 2020. *Taverna.* Retrieved July 20, 2020 from https://taverna.incubator.apache.org/
[5] Daniela Berardi, Giuseppe De Giacomo, and Massimo Mecella. 2005. Basis for automatic service composition. *Tutorial at WWW* 2005 (2005).
[6] Daniela Berardi, Giuseppe De Giacomo, Massimo Mecella, and Diego Calvanese. 2005. Automatic composition of process-based web services: a challenge. In *Proc. of the WWW*, Vol. 5. 20.
[7] P. Bourque and R.E. Fairley. 2014. Guide to the Software Engineering Body of Knowledge, Version 3.0. www.swebok.org IEEE Computer Society.
[8] Antonio Bucchiarone and Stefania Gnesi. 2006. A survey on services composition languages and models. In *International Workshop on Web Services–Modeling and Testing (WS-MaTe 2006)*. Citeseer, 51.
[9] Loli Burgueño, Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastien Mosser, Richard F Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, et al. 2019. Contents for a Model-Based Software Engineering Body of Knowledge. *Software and systems modeling* 18, 6 (2019), 3193–3205.
[10] Camunda. 2020. *Camunda BPM.* Retrieved July 20, 2020 from https://camunda.com/
[11] N. Diakov and F. Arbab. 2004. Compositional Construction of Web Services Using Reo. In *Proceedings of the 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure.* 49–58.
[12] Claudio Guidi, Roberto Lucchi, Roberto Gorrieri, Nadia Busi, and Gianluigi Zavattaro. 2006. SOCK: a calculus for service oriented computing. In *International Conference on Service-Oriented Computing*. Springer, 327–338.
[13] Marzieh Hamzei and Nima Jafari Navimipour. 2018. Toward efficient service composition techniques in the internet of things. *IEEE Internet of Things Journal* 5, 5 (2018), 3774–3787.
[14] Alexis Huf and Frank Siqueira. 2019. Composition of heterogeneous web services: A systematic review. *Journal of Network and Computer Applications* 143 (2019), 89–110.
[15] Anneke Kleppe. 2008. *Software language engineering: creating domain-specific languages using metamodels.* Pearson Education.
[16] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. 2015. Web Service Composition: A Survey of Techniques and Tools. *ACM Comput. Surv.* 48, 3, Article 33 (Dec. 2015), 41 pages. https://doi.org/10.1145/2831270
[17] Nikola Milanovic and Miroslaw Malek. 2004. Current solutions for web service composition. *IEEE Internet Computing* 8, 6 (2004), 51–59.
[18] Fabrizio Montesi. 2020. *Chor.* Retrieved July 20, 2020 from http://www.chor-lang.org
[19] Netflix. 2020. *Netflix Conductor.* Retrieved July 20, 2020 from https://netflix.github.io/conductor/
[20] OASIS. 2020. *BPEL.* Retrieved July 20, 2020 from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.htm
[21] OMG. 2020. *BPMN 2.0.* Retrieved July 20, 2020 from https://www.omg.org/spec/BPMN/2.0/
[22] Hye-young Paik, Angel Lagares Lemos, Moshe Chai Barukh, Boualem Benatallah, and Aarthi Natarajan. 2017. *Web Service Composition: Overview.* Springer International Publishing, Cham, 149–158. https://doi.org/10.1007/978-3-319-55542-3_5
[23] M. D. Preda, S. Giallorenzo, I. Lanese, J. Mauro, and M. Gabbrielli. 2014. AIOCJ: A Choreographic Framework for Safe Adaptive Distributed Applications. In *International Conference on Software Language Engineering*. 161–170.
[24] Michael Rosen, Boris Lublinsky, Kevin T Smith, and Marc J Balcer. 2012. *Applied SOA: service-oriented architecture and design strategies.* John Wiley & Sons.
[25] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. 2014. Web services composition: A decade's overview. *Information Sciences* 280 (2014), 218–238.
[26] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. 2003. Workflow Patterns. *Distributed and Parallel Databases* (July 2003), 5–51. https://doi.org/10.1023/A:1022883727209
[27] W3C. 2020. *WS-CDL.* Retrieved July 20, 2020 from https://www.w3.org/TR/ws-cdl-10/
[28] WSO2. 2020. *Ballerina.* Retrieved July 20, 2020 from https://ballerina.io/
[29] Elyas Ben Hadj Yahia. 2017. *A language-based approach for Web service composition.* Ph.D. Dissertation.
[30] YAWL. 2020. *Yet Another Workflow Language.* Retrieved July 20, 2020 from https://yawlfoundation.github.io/
[31] Zeebe. 2020. *Zeebe.* Retrieved July 20, 2020 from https://zeebe.io/what-is-zeebe/
[32] Önder Babur, Loek Cleophas, and Mark van den Brand. 2019. Metamodel clone detection with SAMOS. *Journal of Computer Languages* 51 (2019), 57 – 74.