ADASERC: Advanced Analysis of Service Compositions

Special Session at SERVICE COMPUTATION 2017
The Ninth International Conferences on Advanced Service Computing
February 19 - 23, 2017 - Athens, Greece

Thomas M. Prinz
Course Evaluation Service
Friedrich Schiller University Jena, Germany
e-mail: Thomas.Prinz@uni-jena.de

***Abstract*—The paradigm of service-oriented programming breaks through the practically proven lifecycle of programming paradigms. While the history of those paradigms shows that it is necessary to provide a programming language fundamentally based on the concepts of the new paradigm, such a complete and accepted language is missing in the context of service-orientation. However, we need such a language to build compilers, algorithms, and analyses, subsequently, to support the development and to avoid failures as early as possible. A service-oriented programming language should be able to distribute objects and functionality along a network but, simultaneously, it should hide that complexity from the developer. It also should allow for external and asynchronous service calls without breaking a well-structured program code. Those and further fundamental basics arise when scratching the surface of the service-oriented paradigm. Therefore, currently existing service compositions and systems should be analyzed to extract the requirements for a service-oriented programming language. Based on those requirements such a language can be implemented and, for this, support can be developed. A special track on *Advanced Analysis of Service Compositions* was held as part of the SERVICE COMPUTATION 2017 conference in Athens, Greece, in which such analysis results were provided and considered.**

Keywords–Analysis, service, composition, programming, language.

I. INTRODUCTION

Regarding the history of programming paradigms in computer science, new programming paradigms seem to always have a similar lifecycle based on five steps explained in the following. At first, (1) a new paradigm evolves evolutionary from an existing ancestor programming paradigm. For example, the paradigm of object-oriented programming is based on concepts of procedural programming languages and it is, therefore, its evolutionary successor (cf. the history of programming languages as a graph [1]).

After this birth of a new programming paradigm, (2) either existing languages are extended to fit the new concepts or completely new languages appear. Such new programming languages support the new paradigm per design. As existing programming languages change and new languages are born, (3) new compilers have to arise to translate the programs written in those languages to (virtual) machine-understandable code. It is common to transform concepts of a new paradigm to concepts of its ancestors. For example, most object-oriented programming languages will be transformed to procedural-like code which is then translated to an assembler language. This makes it possible to reuse existing compilers and compiler algorithms.

In these years of increasing compiler technology, (4) first programs are built with the new techniques followed by medium-sized software and ending in very complex systems. Although new programming paradigms facilitate writing good code and implementing large, well thought-out systems in many cases, a great deal of failures appear during runtime and cause high costs. For this reason, eventually, (5) the tool support for the development of applications with the new paradigm increases. This tool support includes methodical approaches (like class diagrams for object-oriented programming languages) as well as practical theories like integrated development environments (IDEs) with their immediate and detailed failure feedback.

"Those who cannot remember the past are condemned to repeat it." [2, p. 284] This quote of the philosopher Santayana should to remind us to consider the history to avoid failures of the past in the future. However, the evolution of the *service-oriented* programming paradigm seems not to pass the same established lifecycle like many other programming paradigms before. Although this paradigm must go through the first step of evolutionary emergence, there is no established new programming language, which is fundamentally based on the concepts of the service-oriented programming paradigm.

The *Web Service Business Process Execution Language* (WS-BPEL) [3] was a first step to a real service-oriented programming language. However, with its XML syntax, it was decreased to a hard to implement standard never getting the full support of other programming languages. If we compare the trend for WS-BPEL with the trend of other methodical service-oriented languages like the *Business Process Model and Notation* (BPMN) [4], WS-BPEL has a sinking priority in the last four years [5]. Though *BPMN* was created to describe abstract language business processes, its syntax cannot be seen as a complete programming language.

Since this important step, the extension of existing or the arising of new programming languages based on the concepts of the service-oriented paradigm, is not finished yet, the further steps of the lifecycle cannot be started successfully. For this reason, up to now, the service-oriented paradigm is implemented only by frameworks or

workarounds in existing programming languages. For example, *ECMAScript* [6] is able to call services and to order them in a workflow. But the service-oriented paradigm asks for *service orchestration*, *service composition*, and *service selection*. So, the implemented service-oriented paradigm today is a shadow of its possibilities.

II. Programming Language and Environment

In prior work [7], we have considered a service-oriented program, i.e., a service *composition*, and what we need to execute it. The resulting approach is an environment divided into three parts: (1) The *producer side* is the development part of the system consisting of the *service-oriented programming language* and a *compiler*, which transforms the entire program into an *intermediate representation* (*IR*, virtual machine code) and performs some *analyses* [8]. After its compilation, (2) the *IR* is stored within a *service repository* being a file system in the simplest case or a service management system including service discovery, etc. Eventually, on the *consumer side*, a distributed *virtual machine* takes the *IR* from the service repository and executes it [9]. The implementation of such an environment is possible, when a programming language arises containing the fundamental concepts of the service-oriented paradigm.

At that time, we think, it is *absolutely vital* to analyze existing service compositions and to extract the basic requirements on design, syntax, and functionality of service-oriented programming languages. Fundamental concepts like distributed information, external and asynchronous service calls as well as workflows should be considered, naturally. However, there are some more basics to think about, which are not in the focus of research, for example: How should we define the scope of a variable or document within a parallel and asynchronous workflow? How can we hide distributed information from the developer? How can we guarantee the integrity of information? Or in one question: How should we develop a software with a single programming language, while the software can be distributed with all of its information in a network or the world wide web? What should it look like?

As we see, there are fundamental questions which arise when we scratch the surface of service-oriented programming languages. We need advanced analyses of service compositions.

III. First steps

In a special track on *Advanced Analysis of Service Compositions*, which was held as part of the SERVICE COMPUTATION 2017 conference in Athens, Greece [10], first steps towards advanced analyses were considered. The special track got two contributions presenting the need and application for such analyses.

Baumann, Eichhoff, and Roller [11] provide a service composition strategy, which enables the automatic selection of cloud 3D-printers based on certain properties. The selection and composition of 3D-printers seems to be a hard and challenging task since there are a lot of different properties, technologies, etc. to consider. As a solution, the authors propose an ontology to collect the domain-specific knowledge. Within the contribution, this ontology is explained. For this, they analyzed the requirements of 3D-printers as well as the service compositions and evolved a composition framework. It would be interesting to extract the fundamental concepts in the context of a general service-oriented programming language.

The second contribution by Prinz and Amme [12] explains the necessity to explore algorithms and analyses for service compositions to be used later in an IDE. In this context, they show in a case study that the application of accurate analysis techniques considering runtime failures is not suitable for IDEs since accurate analysis techniques result in an inaccurate finding of the roots of failures. It sounds like a paradoxon, but traditional *static* compiler analysis, which consider the program structure and derive abstract information from the program, provide more precise diagnostic information to repair malformed programs.

References

[1] Éric Lévénez, "Computer Languages History," website, visited on January 31th, 2017. [Online]. Available: https://www.levenez.com/lang/

[2] G. Santayana, Reason in Common Sense, ser. The Life of Reason: The Phases of Human Progress. New York: Charles Scribner's Sons, Feb. 1906, vol. 1.

[3] M. B. Juric, B. K. Mathew, and P. Sarang, Business Process Execution Language for Web Services: An Architects and Developers Guide to BPEL and BPEL4WS, second edition ed., ser. From Technologies to Solutions, M. Little and D. Shaffer, Eds. Birmingham, UK: Packt Publishing Ltd., Jan. 2006.

[4] Object Management Group (OMG), "Business Process Model and Notation (BPMN) Version 2.0," OMG, Jan. 2011, standard. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[5] Google Trends, "BPEL, BPMN - Explore - Google Trends," website, visited on January 31th, 2017. [Online]. Available: https://www.google.com/trends/explore?q=BPEL,BPMN

[6] ecma International, "Computer Languages History," ecma International, Jun. 2016, standard. [Online]. Available: http://www.ecma-international.org/ecma-262/7.0/index.html

[7] T. M. Prinz, T. S. Heinze, W. Amme, J. Kretzschmar, and C. Beckstein, "Towards a Compiler for Business Processes - A Research Agenda," in SERVICE COMPUTATION 2015: The Seventh International Conferences on Advanced Service Computing, pp. 49–54.

[8] T. M. Prinz, R. Charrondière, and W. Amme, "Business Processes Compiled — Necessary Support for their Development (Geschäftsprozesse kompiliert - Wichtige Unterstützung für die Modellierung)," in Proceedings 18. Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach am Wörthersee, Austria, pp. 476–491.

[9] T. M. Prinz, "Proposals for a Virtual Machine for Business Processes," in Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015, Jena, Germany, February 19-20, 2015., pp. 10–17.

[10] International Academy, Research, and Industry Association (IARIA), "SERVICE COMPUTATION," website, visited on February 1th, 2017. [Online]. Available: https://www.iaria.org/conferences2017/SERVICECOMPUTATION17.html

[11] F. W. Baumann, J. R. Eichhoff, and D. Roller, "Resource Description for Additive Manufacturing — Supporting Scheduling and Provisioning," in SERVICE COMPUTATION 2017: The Ninth International Conferences on Advanced Service Computing, to be published.

[12] T. M. Prinz and W. Amme, "Why We Need Advanced Analyses of Service Compositions," in SERVICE COMPUTATION 2017: The Ninth International Conferences on Advanced Service Computing, to be published.