SmartCLIDE: Shortening the Toolchain of SOA-based Cloud Software Development by Automating Service Creation, Composition, Testing, and Deployment

Theodore Maikantis

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, teomaik19@gmail.com

Theodore Chaikalis

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, chaikalis@uom.edu.gr

Apostolos Ampatzoglou

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, a.ampatzoglou@uom.edu.gr

Alexander Chatzigeorgiou

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, achat@uom.edu.gr

Nowadays the majority of cloud applications are developed based on the Service-Oriented Architecture (SOA) paradigm. Large-scale applications are structured as a collection of well-integrated services that are deployed in public, private or hybrid cloud. Despite the inherent benefits that service-based cloud development provides, the process is far from trivial, in the sense that it requires the software engineer to be (at least) comfortable with the use of various technologies in the long cloud development toolchain: programming in various languages, testing tools, build / CI tools, repositories, deployment mechanisms, etc. In this paper, we propose an approach and corresponding toolkit (termed SmartCLIDE—as part of the results of an EU-funded research project) for facilitating SOA-based software development for the cloud, by extending a well-known cloud IDE from Eclipse. The approach aims at shortening the toolchain for cloud development, hiding the process complexity and lowering the required level of knowledge from software engineers. The approach and tool underwent an initial validation from professional cloud software developers. The results underline the potential of such an automation approach, as well as the usability of the research prototype, opening further research opportunities and providing benefits for practitioners.

Development frameworks and environments • Development frameworks and environments• Software development process management

Additional Keywords and Phrases: Service-Oriented Architecture, Cloud Software Development, Automated Software Engineering

ACM Reference Format:

T. Maikantis, T. Chaikalis, A. Ampatzoglou, A. Chatzigeorgiou. 2021. SmartCLIDE: Shortening the Toolchain of SOA-based Cloud Software Development by Automating Service Creation, Composition, Testing, and Deployment. In PCI '21: 25th Pan-Hellenic Conference on Informatics, 26 - 28 November 2021, Volos, Greece.

1. INTRODUCTION

Service Oriented Architecture (SOA) aims to enable the quick and easy creation of software applications, or business processes (composed of services) that are accessible through a network [6]. To this end, an application using SOA is composed of several micro-services, each fulfilling a specific functionality [8]. Since each service represents a single functionality in a business process, the composition can be considered as a sequence of steps towards the completion of an overall goal [5]. In the SOA context, the micro-services are considered as black-box software components that are completely reusable via a standard-based interface over the network [6]. Recent growth indicates that low-code technologies are likely to be used and adopted by several types of companies, for internal and industrial use [1]. As a result of the popularity and usefulness of SOA practices, a high demand for the creation and composition of services has been observed.

In order for developers to develop service-based cloud applications in an efficient and productive manner, they are expected to have a broad spectrum of skills, mainly focusing on the fluent use of various tools and technologies. In particular, to deliver a working system of acceptable quality, the software engineer must possess knowledge and skills on how to configure and use several technologies, many of which are new and obscure for a novice developer. Despite the need for knowledge covering the end-to-end development spectrum, each individual technology (most of the times) influences a single aspect of the development process, but nevertheless requires a significant

amount of time, effort and knowledge to configure. Even if a developer does not possess the necessary skills, he / she is usually forced to acquire them under time pressing conditions, leading to errors and compromises in the quality of both the process and the end product.

Below we present a standardized process that relies on several well-established steps that are mandatory for the development of service-based software application for the cloud:

- ***Service Composition*** is considered as the integration of multiple services in a black-box fashion. The developer has to decompose a problem into smaller tasks which, in turn, are implemented as services. They then have to structure the problem using the appropriate architecture, in order for the end result to be functioning correctly and of good quality.
- ***Service Discovery*** is the search for a service suited for the needs of a particular task. This process can be quite cumbersome as it requires an extensive manual search by the developer. The developer has to go through several service registries' documentations in order to find a suitable, working and accessible service.
- ***Service Creation*** is the development of a new service from scratch. This is done for the sake of service composition and in case the discovery process fails. During the creation process, the developer implements the required functionality of a task by developing new code and coding the corresponding endpoint.
- ***Testing*** corresponds to the validation of a created service and the service composition as a whole. Testing and quality evaluation is integral to the lifecycle and usefulness of an implementation, as better-quality software tends to make better use of the invested time and funds dedicated to its development. There are several levels of testing and accessing quality, such as unit, integration and system testing while the performance and availability of cloud-based services should also be continuously monitored.
- ***Deployment*** can be considered the final step of the development, where a service instance is made available through a network after being deployed through one or more hosting models such as platform as a service (PaaS) and/or infrastructure as a service (IaaS). It is an important aspect of the process as it requires specialized knowledge in order to leverage the cloud, and in order to achieve a dynamic runtime configuration offering performance and scalability.

Each stage of the development contains several non-trivial and time-consuming development and configuration steps. As a result, a manual approach is prone to errors that can lead to non-optimal architectures and non-performant services, which do not take the full advantage of the cloud infrastructure.

The goal of this paper is to present a tentative solution to this problem, (namely SmartCLIDE[1]) that automates cumbersome and complicated tasks by abstracting out technical configuration details that are forced by existing IaaS and PaaS technologies. SmartCLIDE is an EU funded project that aims at the provision of a cloud-based IDE that will guide the development of cloud applications, using SOA and the low-code programming paradigm. The proposed solution has received preliminary evaluation by professional cloud software developers.

2. RELATED WORK

There is an adequate amount of work investigating the usefulness of low-code approaches and the difficulties that are frequently encountered in new application production. By the term 'Low-code development' we refer to the ability of implementing software solutions through the reuse of existing building blocks that developers can assemble into processes/workflows. Low-code development shows great potential in the software development industry. This was attributed to lack of available programmers, as well as, to the increasing frequency of introducing changes in IT systems. By employing low-code programming the aforementioned problems are alleviated, because the labor-intensive programming phase is not necessary to build and maintain applications [7]. Similarly, companies and organizations that depend on code are concerned about the cost of quality documentation and training of new human resources in development [4]. Another research, investigated the most common issues encountered while using LCSD (Low-Code Software Development) platforms. Among those, it was found that LCSD developers found testing, challenging for their applications, thus making debugging harder. Another issue mentioned as challenging and problematic, was the integration of external APIs of services [1]. By making development easier

[1] . https://smartclide.eu/

and combining it with low-code practices, we can facilitate application creation, while maintaining and enhancing the ability to develop code and services from scratch in order to satisfy newly emerging required functionalities.

3. SMARTCLIDE SERVICE CREATION, COMPOSITION, AND TESTING

In this section, we present the proposed approach for Service Creation, Composition, and Testing (see Section 3.1), as implemented in the SmartCLIDE IDE (see Section 3.2). Our toolchain aspires to deliver a faster, easier and more user-friendly solution compared to the from-scratch development of SOA-based cloud applications. The approach targets to automate several steps and easily provide functionalities to the developer, in a manner that enhances productivity and minimizes distractions and time-consuming interactions with external tools.

3.1 SmartCLIDE Methodology Overview and Selected Technologies

The SmartCLIDE development process builds upon the generic steps defined in Section 1. The proposed process (see Figure 1) starts with the decomposition of the targeted application to smaller steps (creating a workflow) in a visualized way. Apart from the decomposition, the software engineer sets the basic constraints (such as quality attributes of interest, programming language, etc.). Next each step of the workflow is specified and subsequently SmartCLIDE attempts to discover a service that can fulfil the task functionality. If a fitting service is identified then it is mapped to the task, if not, a new service should be created for fulfilling the task. Upon the mapping of all tasks to existing or new services, the workflow is tested and then deployed.

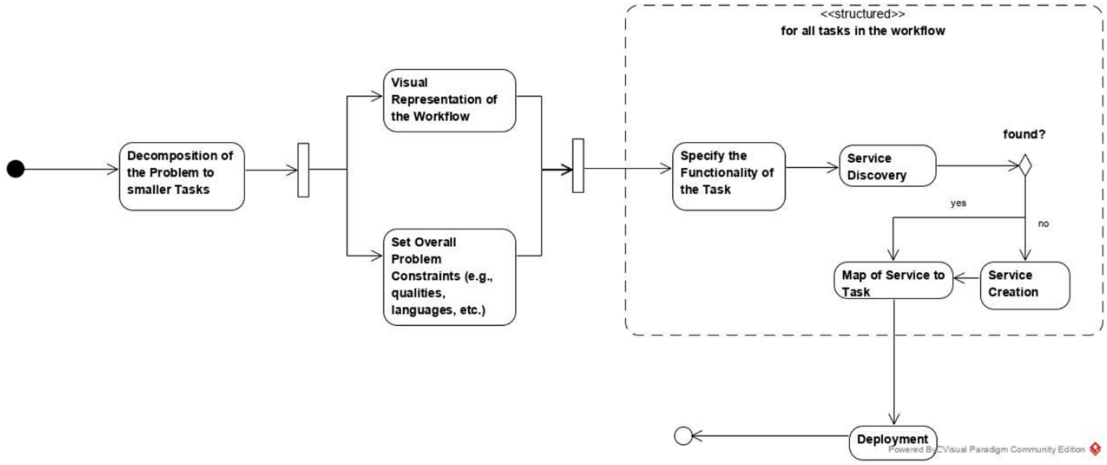

Figure 1 The SmartCLIDE Service-based Cloud Application Development Process

Having the high-level process defined, each step needed to be implemented, based on commonly used technologies in the cloud development industry. Implementing the aforementioned end-to-end solution requires the development and combination of several tools and functionalities. Below we present our main decisions one-by-one, organized by the basic steps defined in Section1.

Service Composition: Following the low code approach, there is a need for a popular yet easy to use framework to support the composition of services. This framework should allow the design of composition scenarios in a graphical, canvas-based format with services provided as artifacts that can be integrated in a drag-n-drop manner, even by non-technical users. To this end, we opted for the Business Process Model Notation (BPMN[2]) as the modelling container that will support all the requirements. Due to its nature, BPMN offers an ideal solution for composing heterogeneous services from different sources so that cumulatively serve a common purpose and deliver business value. This can be achieved by treating created services as executors of BPMN service tasks and combine them in a structured manner by creating connections in an interactive drawing canvas. In the context of a modern, cloud-based environment the most profound solution of service task implementation is to be in the form of Remote Web Service. This solution enables seamless extensibility of the platform in the sense that the addition of new service tasks requires only the definition of the endpoint and data contract.

[2] https://www.bpmn.org/

Service Discovery: During the composition process, there is the need to fulfill task functionalities by software implementations. This can be done by pairing tasks with existing black-box services. Such services need to be discovered, usually though a manual search of the developer. In SmartCLIDE service discovery is assisted by an AI agent that receives as input necessary functional information about the service that is going to be used, and goes through several sources (e.g., programmable web) for identifying fitting services [2], [3].

Service Creation: In case the service discovery process does not yield any satisfactory results, the developer is forced to implement the needed functionality as a new service from scratch. By analyzing the required steps to develop a fully functional service, we came up with an approach to automate service creation. The approach considers the main infrastructure-related steps that are required to setup before starting implementing the service: (a) Create a code repository; (b) Create a CI/CD pipeline, e.g., GitLab CI/CD or a Jenkins job; and (c) Configure them to interact with each other. Next, during code development per se, the developer needs to interact with several tools:

- for writing code
- for collaborative development
- for creating, performing tests and verifying the results
- for evaluating the quality and maintainability of the code
- for monitoring the progress of the project
- making the service invokable by being packaged into an image

To accommodate and automate these features, a complete toolset was implemented, to hide the complexity of configuring and executing the aforementioned tools. The result adopts a one-stop shop philosophy where tools are configured once while the necessary functionality is automatically invoked through API calls.

Service Testing: Testing plays a crucial role for software development. It ascertains a service's correct operation but also aids towards a faster and more structured way of coding. In our context as with everything else, testing needs to be automated to a degree and made easy, so as to conform to our low code – low skill requirements doctrine. Additionally, there needs to be testing support available for individual services, but also for composition of services, all embedded and used through a CI/CD pipeline. Finally, as software quality and maintainability are important for the lifecycle of a product, care must be taken to ensure that the user is notified about its progress.

Service Deployment: Finally, after the completion of the composition and / or creation of services, the final product should be able to be deployed without the knowledge of complex cluster technologies. Again, a testing process can be repeated in order to check for the correct integration and function of the deployed final result. As a first step, in the SmartCLIDE project, we deal with the deployment of individual services, i.e. how to deploy containers after pulling an image from a registry. In later stages of the project, deployment of services to a swarm and container orchestration will be investigated.

3.2 Tool Support

In the context of SmartCLIDE we have considered only mature, open-source implementations that are easily extensible and support cloud hosting. To this end we researched available tools and technologies, evaluated them based on our needs and verified that the licensing scheme allowed modifications with zero restrictions. Apart from the existing software, we developed in-house source code for all internal modules of the SmartCLIDE solution. The source code is available online in the Eclipse Research Labs repository[3].

Workflow Development - Service Composition: The subsystem for workflow creation lies in the heart of SmartCLIDE solution with obvious implications in the stability of the overall architecture. Therefore, in an attempt to enhance future maintainability and minimize failure risks we opted for JBPM[4] which is a wide-adopted solution

[3] https://github.com/eclipse-researchlabs/smartclide-service-creation
https://github.com/eclipse-researchlabs/smartclide-service-creation-theia
https://github.com/eclipse-researchlabs/smartclide-TD-Reusability-Index
https://github.com/eclipse-researchlabs/smartclide-td-reusability-theia
https://github.com/eclipse-researchlabs/smartclide-TD-Interest

[4] https://www.jbpm.org/

for process workflow development and automation. JBPM is a complete process automation toolkit that supports features for the whole lifecycle of a business process, from design and testing to execution and monitoring. All features can be executed by the end-user through a dashboard for process authoring and administration called Business Central. The development and testing of business process diagrams is performed by using the online editor, which supports drag-n-drop functionality for the addition of BPMN artifacts such as User Tasks, Start/End nodes, and Decision Gateways. An example of a business process with composition of several remote Web Services from different sources is depicted in Figure 2, along with a design proposal for an enhancement of the remote service usage on the editor canvas. Current approach dictates the usage of a generic web service task that can be parameterized by configuring the remote endpoint and providing input for service security (username, password or auth token), parameters or any other required data, depending on the service contract. However, this process loads the designer with unnecessary effort as it involves exploration of the documentation of every employed service. To this end, the solution wireframe of Figure 2, proposes a new toolbox that will display all available services as tool artifacts in the right toolbox that can be dragged on the design canvas as remote services, with predefined data contract and special input requirements, significantly facilitating the job of the process designer.

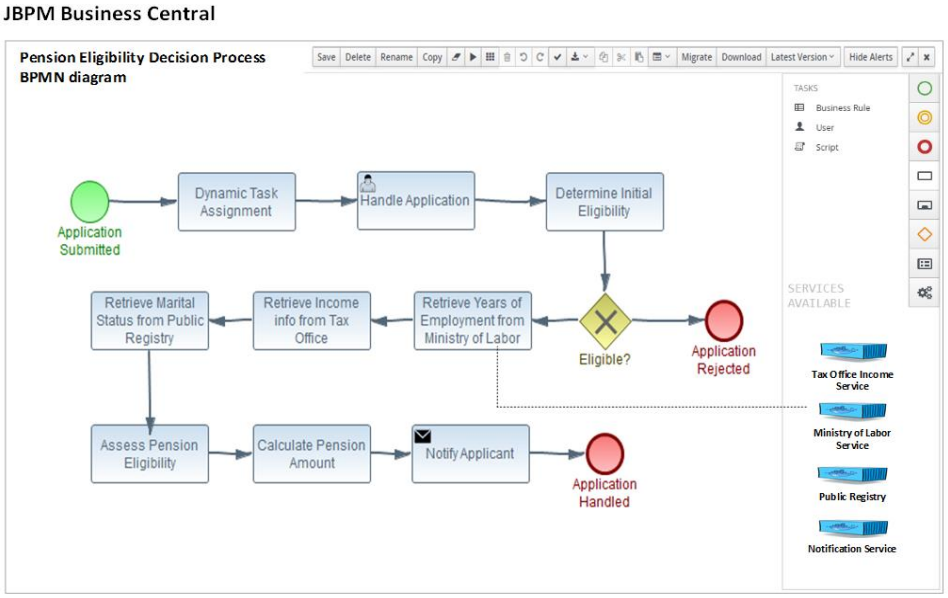

Figure 2. The JBPM editor canvas, displaying an example process. The right toolbox area displays a mockup design proposal for the offering of existing services as drag-n-drop artifacts.

Finally, the core module of JBPM framework is the process execution engine, also known as "kie-server", responsible for running process instances, maintaining state, and offering a wide range of API endpoints that expose several functionalities and offer seamless extensibility.

Service Discovery. An important part of automating the development process is the discovery of services. Service discovery will be provided by the SmartCLIDE platform and is based on meta-data on source code. After the user sends a query with the service specification, through a Natural Language Query Interface, the discovery will draw results from web pages, code repositories and Service Registries by invoking 3rd party search APIs. SmartCLIDE can rewrite the provided user input query on the basis of indexed popular search queries leveraging AI-based techniques, displaying the results of identified services to the user as a ranked list [2], [3].

Service Creation: Considering the actual coding procedure as the central point of software development, the main UI for service creation is the IDE (see Figure 3). Several code development tools were evaluated, and Eclipse Theia was selected as the final solution. Following our previous way of thinking, each service creation functionality is accessible through the Eclipse Theia IDE to minimize distractions and make the whole development process easier and more efficient. Several extensions were created for the Theia platform to provide a level of automation and abstraction for technical configuration details. The extensions come paired with back-end components that

handle requests and perform the necessary configuration tasks by leveraging available tool APIs, for example the creation of a GitLab repository and its pairing with a Jenkins or GitLab CI/CD pipeline.

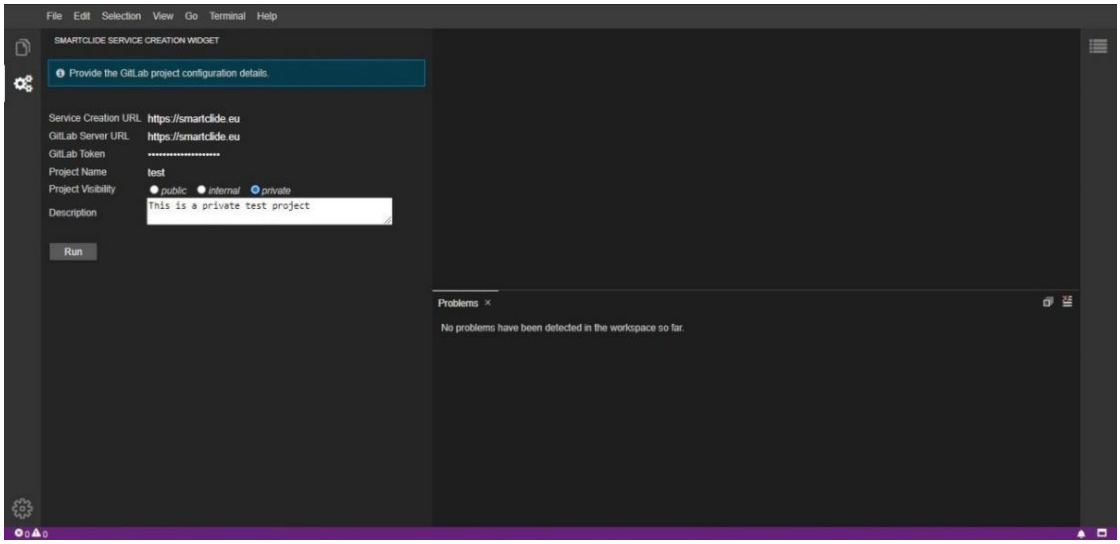

Figure 3. Service creation Theia extension

Service Testing: By adhering to the OpenAPI Specification using Swagger and Selenium, we have the ability of automatically generating unit and integration tests. Running test from the IDE, and getting the feedback on integration and test coverage, as well as the test cases that were performed, is vital to the debugging and error detection of a project. Apart from isolated service testing, we can also test the complete process workflow integration by leveraging the internal testing features provided by the JBPM Business Central as described in previous sections.
Furthermore, to safeguard the quality of the final resulting service, we developed several back-end services that perform analysis on code and calculate quality metrics such as Technical Debt (TD) Interest, TD Principal, and Reusability Index (see Figure 4). These tools perform static source code analysis; assess its level of Technical Debt, computer several source code metrics and examine the entire history of commits within the corresponding git repository. These tools are abstracted away from the developer who is presented only with selected indicators highlighting key pieces of information, as shown in Figure 4.

Service Deployment: Continuous Integration and Deployment are vital to modern microservice-based applications, commonly related to containerization practices. In such a case, the CI/CD workflow must support the automation of image build, uploading to proper repositories, and also remote deployment by using orchestration platforms such as Kubernetes or other cloud management systems such as OpenShift. SmartCLIDE supports CI/CD integration with Kubernetes and as a result it is possible to handle deployment with pipeline scripts. Furthermore, the ability to keep containers and images up-to-date is available, through plugins such as watchtower. However, CI/CD configuration may get complex, especially when considering that different CI/CD engines could be used by the same developer leading to potential differences in syntax (e.g., configuration files). Thus, a more robust and versatile way of deployment needs to be researched.

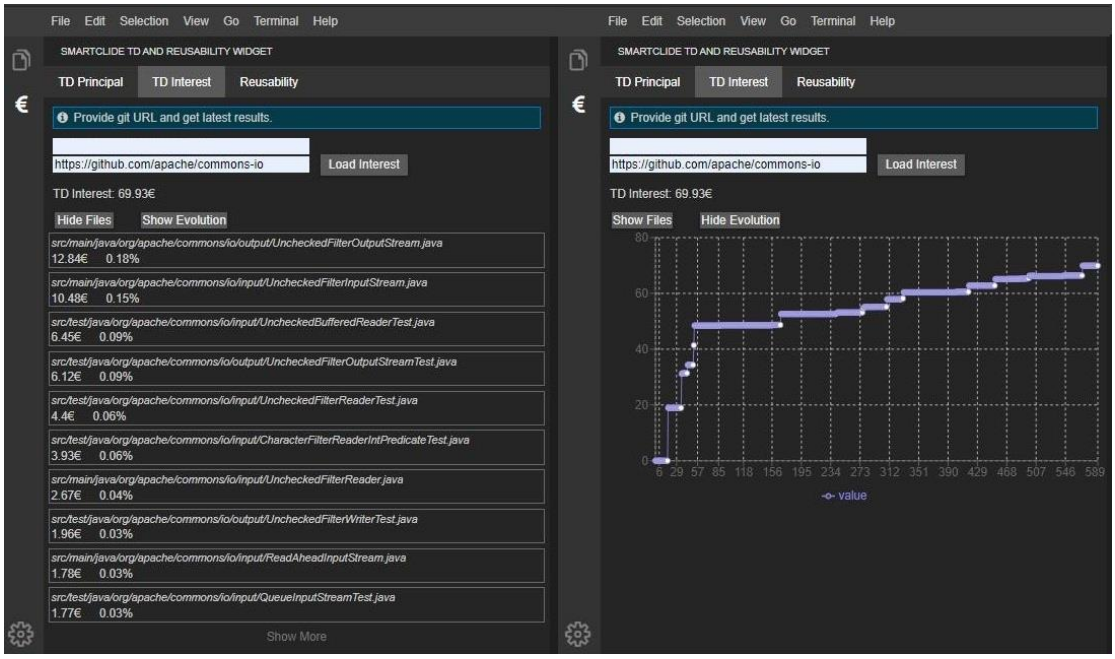

Figure 4 Technical Interest Theia extension

4. EXPECTED BENEFITS AND INITIAL EVALUATION

The proposed solution offers a complete toolset for composing, creating, testing and deploying services, by combining state-of-the-art technologies and tools. The expected benefits are presented using a visual representation illustrating the 'before' and 'after' processes of cloud-based service development. From Figure 5 it becomes obvious that before SmartCLIDE (left), in order to create a new service, the developer had to manually interact with and configure four different tools (git, Jenkins, IDE, and Docker). On top of that, by also considering the composition of services, the discovery of services and the deployment of services, we can see that the time, effort and knowledge required could increase dramatically. On the other hand, in the SmartCLIDE solution (right), the user only interacts with the Eclipse Theia IDE. All complicated functionalities and configurations are hidden from the user, who interacts with our backend component through Theia extensions. Consequently, the user's focus remains on the code development and not on managing and interacting with all the necessary tools. Additionally, the time consuming and error prone tool configurations are abstracted away from the development process. Finally, all the essential information is retrieved by the Service Creation component, which then redirects it to the Eclipse Theia front-end, where it is presented to the user. As a result, the entire development process is less error prone, but also more efficient and developer-friendly.

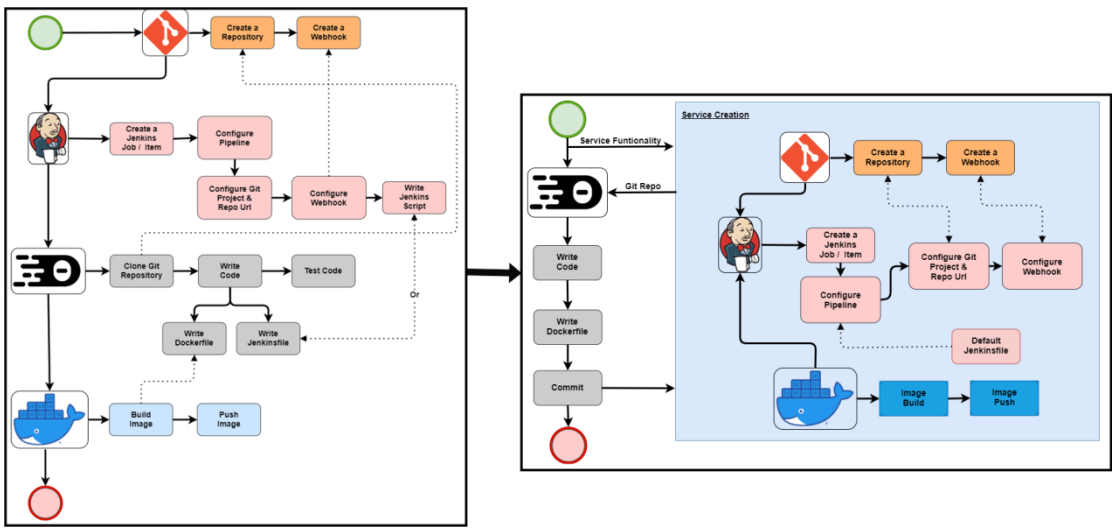

Figure 5 Service Creation – Before SmartCLIDE (left) & after SmartCLIDE (right)

While the proposed toolkit has not been formally evaluated due to time limitations and immaturity of some steps, we have sought feedback on the developed prototype by three (3) professional software developers with experience on microservice-based enterprise applications. Following a two-hour presentation of the corresponding tools and Q&A session, we asked the developers to experiment with the prototype tools, mimicking the process of developing, building, testing and deploying a new service. We asked the developers to provide an estimate of the additional or less time to complete the relevant tasks (compared to their typical work scenario) and to record any flaws, limitations or suggestions for improvement. The obtained feedback is quite encouraging in the sense that the developers acknowledge the need for such automation and anticipate a reduced development time, in case all steps get automated. They stressed the need to familiarize with the new toolkit and the importance of providing extensive tutorials and 'how-to' videos. The ability of searching and discovering services through semantic information would be highly welcome including guidance on how to orchestrate clusters of containers as in the case of Kubernetes for which the learning curve is rather long. The overall user experience was highly rated by the developers; nevertheless, there is room for improvement in illustrating the next user step as in the case of a wizard.

5. CONCLUSION

Service-based software architectures are widely advocated as an ideal alternative for modernizing monolithic legacy applications and for building new, agile and extensible cloud-based enterprise systems. While the architectural advantages of services have been widely studied, programming, composing, creating, testing and deploying services remains a non-trivial task requiring expertise, a lot of experimentation and familiarity with a number of diverse technologies and tools. In this paper we propose an approach for facilitating cloud-based software development by extending a well-known cloud IDE from Eclipse. The end goal is to shorten and simplify the toolchain for cloud development, hiding the underlying complexity from software engineers.

ACKNOWLEDGMENTS

Work reported in this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871177 (project: SmartCLIDE).

REFERENCES

[1] Md Abdullah Al Alamin, Sanjay Malakar, Gias Uddin, Sadia Afroz, Tameem Bin Haider, & Anindya Iqbal. (2021). An Empirical Study of Developer Discussions on Low-Code Software Development Challenges.

[2] Alizadeh-Sani, J., Martinez, P. P., Gonzalez G. H., Gonzalez-Briones A., Chamoso, P., Corchado, J.M. (2021). A Hybrid Supervised/Unsupervised Machine Learning Approach to Classify Web Services. In Highlights in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection (pp. 93–103). Springer International Publishing.

[3] Berrocal-Macias, J., Alizadeh-Sani Z., Pinto-Santos F., Gonzalez-Briones A., Chamoso P, Corchado J. M. (2021). Services Extraction for Integration in Software Projects via an Agent-Based Negotiation System. In Highlights in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection (pp. 241–252). Springer International Publishing.

[4] H. A. A. Chaudhary, T. Margaria, "Integration of micro-services as components in modeling environments for low code development", Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE), May, 2021.

[5] Groves D, Successfully planning for SOA, BEA Systems Worldwide, 11 Sept 2005

[6] Zaigham Mahmood. 2007. Service oriented architecture: potential benefits and challenges. In Proceedings of the 11th WSEAS International Conference on Computers (ICCOMP'07). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 497–501.

[7] Robert Waszkowski (2019). Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine, 52(10), 376-381.

[8] Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design - An interdisciplinary modeling approach for SOA projects. Technical article, IBM (2 June 2004). Online: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/