The Importance of Low-Coding Solutions for Service Engineering in Smart Regions: A Case Study

Rubén Ruiz-Torrubiano, Gerhard Kormann-Hainzl, Antonino Rossi, Cristian Knebel, Egor Evlampiev

IMC University of Applied Sciences Krems, Austria

Abstract. Smart service systems are one of the enabling technologies of the smart region. These are software systems capable of learning, dynamic adaptation and decision making based upon received and transmitted data. By using smart service systems, the stakeholders in the smart region (government, citizens and enterprises) can harness large volumes of data generated by Internet-of-Things (IoT) infrastructure, including sensors and other digital devices. However, developing smart service systems remains largely a problem delegated to conventional software engineering techniques, which can be costly for a single community or region and cannot be easily shared between different regions. In this paper, we focus on the question of whether low-coding solutions can foster the development of smart services in smart regions, and how they can be implemented. We show a case study (points of interest) that illustrates the use of a particular tool developed for this purpose that we call Sagittarius, and generate and deploy a smart service that meets the case requirements.

Keywords: Smart services, smart regions, low-coding solutions.

Beitrag im Rahmen des 16. Forschungsforums der österreichischen Fachhochschulen von 19.-20. April 2023 an der FH St. Pölten.

1 INTRODUCTION

Service systems can be defined as dynamic value-cocreation configurations of resources, connected internally and externally to other service systems by value propositions [1]. This configuration includes in general people, organizations, shared information and technology. From the point of view of service science, a service denotes "the application of specialized competences (operant resources: knowledge and skills), through deeds, processes, and performances for the benefit of another entity or the entity itself" [2]. The concept of value-cocreation [3] plays a central role for service systems: service consumers and service providers interact by creating value in the form of a value proposition, which is an invitation from actors to one another to engage in service [4]. The combination of these concepts with advances and developments in information technology (IT) results in the emergence of smart service systems (SSS), which are software systems capable of learning, dynamic adaptation, and decision making based upon received and transmitted data [5]. The actual connection between service providers and consumers is realized by means of smart products. These are physical products with networking and data-processing capabilities that enable modeling complex business scenarios in a variety of contexts, like healthcare [6], manufacturing [7] and mobility [8] [9]. In general, smart service systems are capable of monitoring, optimizing and controlling smart products and devices to deliver value for the service participants. In the smart region context, smart service systems enable value-cocreation for the participants of the smart region ecosystem to address its main challenges (like urbanization, climate change, sustainable transport, housing, and healthcare) by an intelligent use of information technologies [10]. One of the main challenges to deliver these goals is an efficient way of engineering smart service systems to enable the communities in a region and its citizens to harness the large volumes of data that are produced by sensors and digital infrastructure to their benefit, and to benefit the communities as a whole [11].

In this paper, we address the question of whether the use of low-coding solutions [12] aids in the development of smart service systems in the smart region context in a significant way. We illustrate the application of a low-coding tool called Sagittarius[1] especially tailored to the smart region context to a particular use case. We compare the engineering process of a smart service for this use case to a hypothetical standard agile software engineering process qualitatively and draw conclusions based on this comparison. This paper is structured as follows: in Section 2, we give a general overview of our methodology. In Section 3, we describe the architecture of Sagittarius and its main building blocks. The use case and the previously mentioned comparison are detailed in Section 4. We conclude with some remarks in Section 5 and give an outlook on future work.

2 METHODOLOGY

In this section we give an overview of the research methodology used in this work. For developing smart services, we follow a general software development lifecycle adapted

[1] https://github.com/IMC-UAS-Krems/Sagittarius

to the smart region context as depicted in Figure 1. This lifecycle is made general enough to accommodate different methodologies like waterfall, agile and other iterative software development methods. We begin by a *use case analysis*, where we collect the general requirements of the use case at hand. In the smart region context, these requirements might include functional and non-functional requirements, such as security and regulatory requirements regarding how data is handled by the system. For instance, if personal data is collected by sensors and used in the context of the service, one requirement might be that data should be anonymized and aggregated so that the original source cannot be tracked anymore.

Figure 1. A general software development lifecycle.

In the *design* phase, the previously collected requirements are used to form a system architecture that can satisfy them from the structural point of view. Usually for this purpose formal languages like UML [13] or semi-formal approaches are used. The result is normally a description of components along with their relationships, data and control flows e.g., as a component diagram. The *implementation* phase encompasses all the activities related to the actual coding and development of the system, alongside integration and unit testing. In the *deployment* phase, the artefacts produced in the implementation part are packaged and executed in a suitable production environment and made public. Typical production environments include computing clouds, which might be public, private or a mixture of both. Finally, the project reaches a *maintenance* state where the deployed artifacts are monitored, errors corrected, and minor features implemented until the end-of-life of the project.

We note that the lifecycle described above is applicable to both classical software development models like waterfall (where these phases are executed sequentially) and iterative models like agile methodologies [14] (where some phases or groups of phases are executed in an iterative way). For instance, design and implementation can be completed in iterations while keeping a linear execution for use case analysis and deployment. In practice, mainly the implementation phase is subject to continuous iterations, while the other phases are iterated sporadically as necessary. Additionally, development of new versions of the service can be accomplished by transitioning from the maintenance to the use case analysis phase, where new requirements are analyzed and refined in a new global iteration of the lifecycle.

Using this lifecycle as a blueprint, we perform a qualitative comparison between a commonly used agile software engineering approach (Scrum [14]) and the usage of Sagittarius for smart service system development. This comparison is based on the following aspects:

- What type of resources are needed, and to which extent? This includes both

personnel and technical resources. Are specialized personnel or hardware needed?

- Stakeholder involvement: How easy is it to integrate the project's stakeholders into the different phases?
- Flexibility: How easily can changes be introduced at any phase in the development process?
- Efficacy: How can it be ensured that the right features were developed in the most straightforward possible way?
- Quality Assurance: How can quality be ensured and maintained?

Note that there are quantitative measures available for all the previously mentioned dimensions, but we choose a qualitative approach since estimating quantities like costs or velocity is generally difficult without setting up an agile development process in practice and make a quantitative comparison, which is out of the scope of the present paper.

3 SAGITTARIUS: A LOW-CODING TOOL FOR SMART SERVICE ENGINEERING

Sagittarius is an open-source low-coding solution currently under development that is especially designed for the smart region context. For this tool, we chose a service-oriented architecture where services interact with each other by means of well-defined interfaces [15]. The definition of smart services in Sagittarius is based on a domain-specific modeling language that we call Smart Service Definition Language (SSDL) [16]. This language is designed to be easily understandable and as near as possible to natural language for reducing entry barriers for users without a technical background. For this purpose, a YAML-like syntax was chosen where four clear aspects of the service can be defined as sections:

- Service metadata: Properties like name or category of the service can be defined here. Also, versioning and other metadata types are supported.
- Data sources: In this section details regarding endpoints and types of data can be defined like type of IoT platform (e.g. Fiware), relevant data fields (depending on the use case) and filtering criteria or data transformations.
- Application: Details on the generated service can be specified in this section, like type of application (web application, smartphone app, etc.) and which visualizations should be included (line charts, scatter plots, maps, etc.).
- Deployment: Additionally, the production environment where the service should be deployed can be specified in this section. This includes the containerization technology to use (e.g. Docker) and the endpoint for the service to be deployed to.

In Figure 2, we give an example SSDL definition for the use case analyzed in Section 4.

```
service:
  name is Point of Interest Barcelona
  version is 1.0.0
  scope is Culture

data:
  - source1:
    name is Points of Interest
    type is Sensor
    provider is Fiware
    uri is https://data.example.at/
    query:
      type is PointOfInterest
    select name, location
```

```
application:
  type is WebApp
  layout is SinglePage
  roles is admin, user
  visualization:
    - Points of Interest Visualization
      type is StreetMap
      area: Barcelona, Spain
      data:
        source1: name, location
```

Figure 2. Definition of smart service in SSDL format for the point of interest use case (see [16] *for details).*

3.1 ARCHITECTURE

Sagittarius is composed of four main services (see Figure 3), which can be listed as follows:

- Web Client: A web application frontend that provides user-facing functionality. The main functionality comprises user login and registration, creation, editing and deploying of services. A browser section will also be available to allow for discovering existing services, aggregate data and adapt pre-existing architectures to one's specific use case.
- API Gateway: This service stands between the web client and all other services. Its use is to redirect requests to the correct service (be it internal or external) while performing authentication and authorization operations[2]. Three endpoints are currently provided:
    - /auth: sign in, sign up and log out.
    - /data: retrieve existing documents, pre-provided templates or find public services.
    - /compile: build a source file, download the final executable or deploy it on the cloud.
- Sagc (Compiler): In this service, the translation between an SSDL definition and a ready to deploy application (the smart service) is performed, alongside all necessary scripts and configuration files (like Dockerfiles, deployment files, etc.). Once an app bundle has been produced, it is possible to either download it and save it for running locally or pass it along to the orchestrator to deploy on the cloud.

[2] Current storage and auth capabilities are being provided by a cloud hosted Supabase instance.

- Orchestrator: This service is responsible for deploying, starting, stopping and upgrading the generated smart services. Deployments can either be docker-based and be displayed for the web or mobile-based to generate mobile-optimized visualizations.

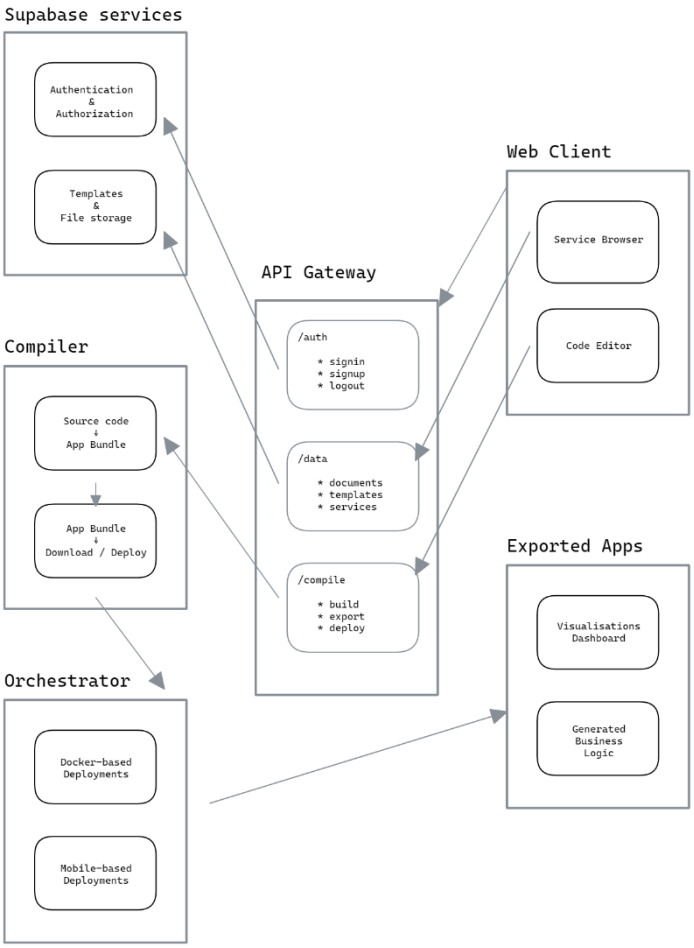

Figure 3. Graphical exemplification of the full architecture and data workflow between different services.

4 USE CASE ANALYSIS: POINTS OF INTEREST

In this section we analyze a concrete use case and qualitatively compare its resolution and implementation using Sagittarius on the one side and Scrum on the other side. We map each step to the software development lifecycle outlined in Section 2 and finally we discuss the results and implications.

4.1 REQUIREMENTS

Points of interest (POI) are labeled locations in a geographic context that provide services to potential visitors. These services can be tourist-oriented (like museums and other attractions) or resident-oriented (like gas stations or electric vehicle charging facilities).

We use a dataset of comprised point of interests from the City of Barcelona in a Fiware instance that can be publicly accessed[3]. The main requirement in this example use case would be to show these points of interest in a map visualization where the user can hover over the highlighted points and see the names of each POI. Therefore, the service should perform the following operations:

- Collect POI data from the relevant IoT platform used.
- Store and process the collected data.
- Provide a user interface to visualize the data (as a map).

4.2 SCRUM PROJECT

We now briefly consider the design, implementation and deployment phases from the point of view of the Scrum agile software development methodology [14]. In the design phase, a first iteration (in some cases known as “Sprint 0”) is done to lay the foundations of the project. In this iteration, basic decisions about technology to use, development frameworks and middleware are made. In general, regular sprint durations like three weeks are preferred, but longer sprints might be used as well. At the end of this sprint, the infrastructure for the project is set up and the development team together with product management has initiated a backlog of tasks that are deemed necessary to complete the project. This would include tasks like analyze the Fiware instance data, create the necessary data structures, build a data access layer, an IoT layer, work on the map visualization and the user management system.

After Sprint 0, the regular implementation phase (see Figure 1) starts and development iterations begin. In each iteration, the Scrum artefacts and rules are followed and iterations are run until the first version of the product is deemed stable. After that, a suitable deployment infrastructure is set up where releases can be published and tested. Finally, the project goes into the maintenance phase where it stays until its end-of-life.

4.3 USING SAGITTARIUS

In contrast, the process for developing the POI smart service using Sagittarius can be described as follows. First, an SSDL definition is written to accommodate the use case requirements. An example of such a definition is shown in Figure 2, where we define metadata, declare data sources and relevant data fields, and finally specify the details of the application like type, user management and visualization type. This can be done both by domain and IT experts and iterates elements from both the design and the implementation phase. Since the architecture of the service is already pre-defined in Sagittarius, there is no need for an explicit architecture definition in this case. The SSDL definition is then compiled so that the code of the application can be generated and tested. For that, we can use a local deployment environment (not shown in Figure 2) so that all stakeholders can participate in a feedback loop. For instance, it could be that the

[3] See https://fiware.github.io/data-models/specs/gsma.html for details

visualization needs more information or some other type of visualization is needed. In that case, the SSDL file would be correspondingly modified and a new iteration would start. This iterative procedure would run until the test phase ends and all stakeholders agree that the service is production ready. As in the previous case, the project goes into maintenance where small changes and minor quality of life updates could be made over time to improve stability or accommodate external feedback.

A graphical representation of the general process for using Sagittarius is given in Figure 4. First, the SSDL definition is written composed of metadata (step 1a), data sources (1b) and application (1c). Then, the “compile” button is clicked and a ready-to-deploy smart service is generated (step 3).

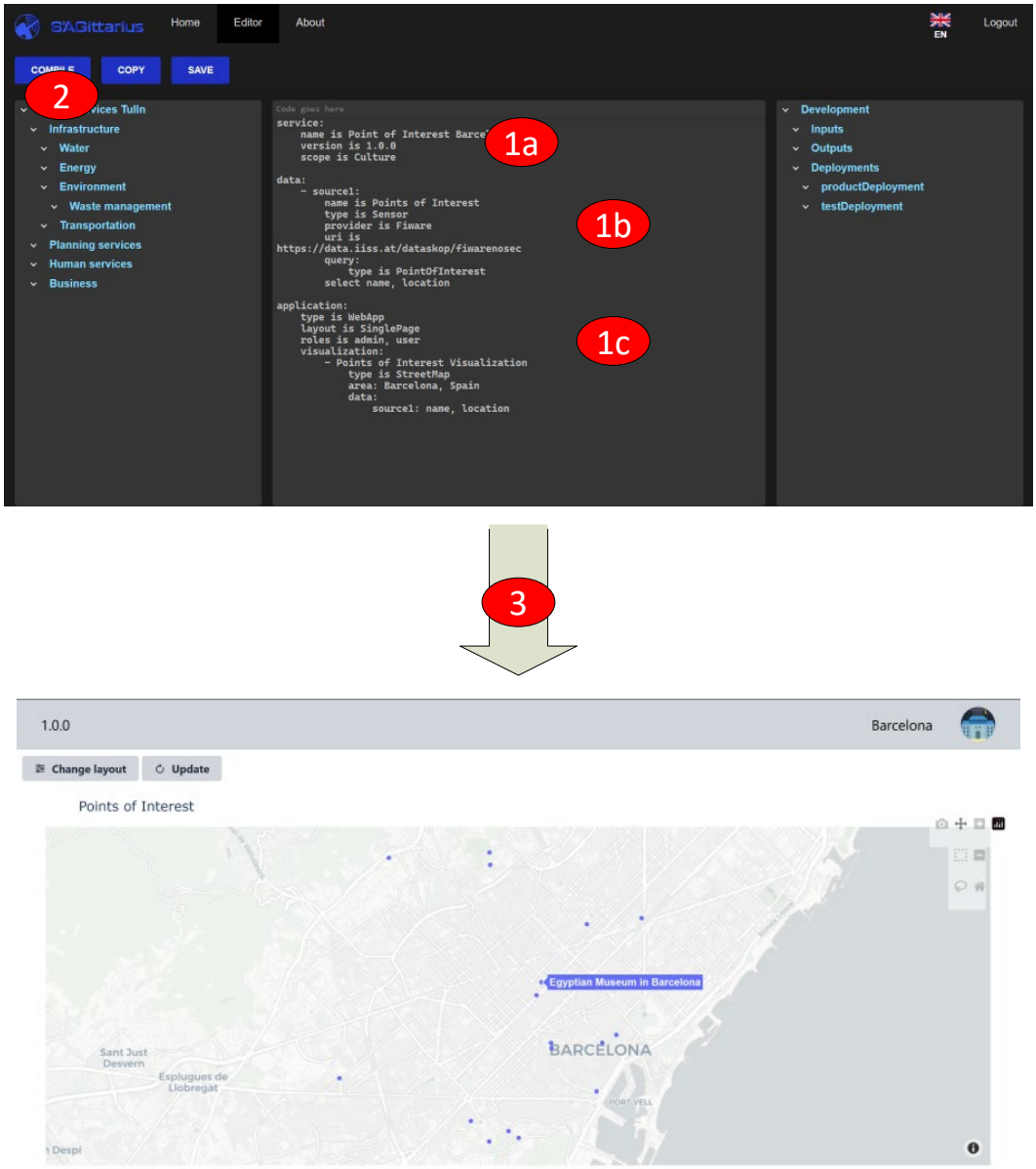

Figure 4. Overview of Sagittarius usage in the development process.

4.4 DISCUSSION

In Table 1 we provide a comparison overview between both methods using the qualitative aspects outlined in Section 2. Regarding which resources are needed, in the case of Scrum highly qualified IT personnel are needed. This includes software engineers, architects, product managers and domain experts with expertise in developing smart services. By contrast, using Sagittarius only domain experts are needed (although we note that these domain experts need to be previously introduced into the SSDL language). While for a standard Scrum development project typical hardware is needed, a Sagittarius project would only need a standard machine with an Internet connection, as Sagittarius is a normal web application. For both methodologies, an appropriate deployment environment (like cloud computing resources) is needed.

In Scrum, stakeholder involvement is achieved by means of the standard Scrum artefacts that include the stakeholders, like Scrum reviews. By using Sagittarius, stakeholder involvement can be continuous, since feedback can be accommodated at any time in the process.

Aspect	Scrum	Sagittarius
Resources	Personnel: highly qualified software engineers and architects and domain experts. Development software, hardware and deployment environment.	Personnel: domain experts. Deployment environment.
Stakeholder involvement	Only through Scrum artifacts (Scrum reviews)	Continuous involvement.
Flexibility	Regular reviews and adaptations.	Reviews and adaptations can be incorporated at any time.
Efficacy	New features/tasks require to go through the Scrum development process.	New features/tasks can be directly incorporated in the tool.
Quality Assurance	Expert knowledge, best practices and security analysis needed.	Best practices already implemented. Integration tests.

Table 1. Overview of qualitative comparison between Scrum and Sagittarius.

The next aspect is flexibility. As mentioned before, adaptations can be incorporated using

Sagittarius at any time. By contrast, in Scrum changes can only be introduced at definite times in each sprint, thus reducing flexibility. Analogously, new features are implemented by executing the involved tasks in the sprint, whereas with Sagittarius new features can be directly incorporated in the tool, since it provides a structured methodology specifically designed for developing smart services. Lastly, quality assurance has to be explicitly integrated into a Scrum project by including testing and security analysis into the development process. On the other hand, Sagittarius already implements best practices in software engineering and security, and therefore only integration and eventually penetration tests are needed. CI/CD, auto-deployment, experimental fuzzing and error-proof pre-provided templates are included to get users and communities easily started.

5 CONCLUSION AND FUTURE WORK

We have introduced Sagittarius, a low-coding tool for developing smart services in the smart region context and applied it to a specific use case. A qualitative comparison with a standard agile development methodology was performed and the discussion suggests that using Sagittarius can result in significant advantages in the aspects mentioned. Especially regarding stakeholder involvement and the need for specialized IT personnel, Sagittarius can help realize smart service projects in communities and regions where these resources are scarce. Moreover, using Sagittarius communities and regions can collaborate and use synergies that could not be harnessed if individual software development processes would be set for each individual project.

Some of the next steps in the development of Sagittarius include native support for public cloud providers and user UI using graphical tools or natural language directly as a pre-step to the SSDL generation phase. Therefore, learning SSDL in the first place should not be needed anymore and the stakeholders can completely focus on the requirements and the domain aspects of the problem at hand.

6 REFERENCES

[1] P. P. Maglio, S. L. Vargo, N. Caswell, and J. Spohrer, “The service system is the basic abstraction of service science,” *Inf Syst E-Bus Manage*, vol. 7, no. 4, pp. 395–406, Sep. 2009, doi: 10.1007/s10257-008-0105-1.

[2] S. L. Vargo and R. F. Lusch, “Why ‘service’?,” *J. of the Acad. Mark. Sci.*, vol. 36, no. 1, pp. 25–38, Mar. 2008, doi: 10.1007/s11747-007-0068-7.

[3] S. L. Vargo and R. F. Lusch, “Service-Dominant Logic: What It Is, What It Is Not, What It Might Be,” in *The Service-Dominant Logic of Marketing*, Routledge, 2006.

[4] J. D. Chandler and R. F. Lusch, “Service Systems: A Broadened Framework and Research Agenda on Value Propositions, Engagement, and Service Experience,” *Journal of Service Research*, vol. 18, no. 1, pp. 6–22, Feb. 2015, doi: 10.1177/1094670514537709.

[5] C. Lim and P. P. Maglio, “Data-Driven Understanding of Smart Service Systems Through Text Mining,” *Service Science*, vol. 10, no. 2, pp. 154–180, Jun. 2018, doi: 10.1287/serv.2018.0208.

[6] H. Khan, K. K. Kushwah, S. Singh, H. Urkude, M. R. Maurya, and K. K. Sadasivuni, “Smart technologies driven approaches to tackle COVID-19 pandemic: a review,” *3 Biotech*, vol. 11, no. 2, p. 50, Jan. 2021, doi: 10.1007/s13205-020-02581-y.

[7] P. Jussen, J. Kuntz, R. Senderek, and B. Moser, “Smart service engineering,” presented at the Procedia CIRP, Elsevier B.V., 2019, pp. 384–388. doi: 10.1016/j.procir.2019.04.089.
[8] D. Beverungen, O. Müller, M. Matzner, J. Mendling, and J. vom Brocke, “Conceptualizing smart service systems,” *Electronic Markets*, vol. 29, no. 1, pp. 7–18, März 2019, doi: 10.1007/s12525-017-0270-5.
[9] R. Dave, N. Seliya, and N. Siddiqui, “The Benefits of Edge Computing in Healthcare, Smart Cities, and IoT,” *JCSA*, vol. 9, no. 1, pp. 23–34, Oct. 2021, doi: 10.12691/jcsa-9-1-3.
[10] A. Wolff, M. Barker, L. Hudson, and A. Seffah, “Supporting smart citizens: Design templates for co-designing data-intensive technologies,” *Cities*, vol. 101, p. 102695, Jun. 2020, doi: 10.1016/j.cities.2020.102695.
[11] M. Dobler, H. Kalkhofer, and J. Schumacher, “Smart Service Development in Public-Private Settings—Assessment Methodology and Use-Cases in the Lake Constance Region,” in *Smart Services Summit*, S. West, J. Meierhofer, and C. Ganz, Eds., in Progress in IS. Cham: Springer International Publishing, 2021, pp. 3–13. doi: 10.1007/978-3-030-72090-2_1.
[12] R. Benac and T. K. Mohd, “Recent Trends in Software Development: Low-Code Solutions,” in *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*, K. Arai, Ed., in Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2022, pp. 525–533. doi: 10.1007/978-3-030-89912-7_41.
[13] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins, “Modeling software architectures in the Unified Modeling Language,” *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 1, pp. 2–57, Jan. 2002, doi: 10.1145/504087.504088.
[14] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile Software Development Methods: Review and Analysis.” arXiv, Sep. 25, 2017. doi: 10.48550/arXiv.1709.08439.
[15] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 3.01 Edition. CreateSpace Independent Publishing Platform, 2017.
[16] Ruiz-Torrubiano, Rubén, Dhungana, Deepak, Kormann-Hainzl, Gerhard, and Paudel, Sarita, “SSDL: A Domain-Specific Modeling Language for Smart City Services,” in *To be published in Smart Services Summit 2022*, Zürich: Springer.