SSDL: A Domain-Specific Modeling Language for Smart City Services

Rubén Ruiz-Torrubiano, Deepak Dhungana, Gerhard Kormann-Hainzl, and Sarita Paudel

Abstract As software services become more and more indispensable in our daily lives, low-code and no-code platforms are gaining significance, especially for non-technical users. We present a novel domain-specific modeling language for the definition of smart service systems in the context of smart cities, which we call Smart Service Definition Language (SSDL). The main goal of this formal language is to provide a basis for a low-coding software engineering approach for the design and deployment of smart services. SSDL is based on the Smart City Ontology (SCO) to provide syntactic and semantic elements related to the smart city environment and aims at defining a syntax as near as possible to human language to overcome acceptance problems for non-technical users. The proposed language can be used by smart system designers and other stakeholders to define the components, actors, data and relationships between the different elements that compose smart service systems in a formal and reproducible way, paving the way for an automatic or semi-automatic generation of ready-to-deploy smart services as independent applications, which may take the form of web services to be integrated in service-oriented architectures. In this paper, we present the syntax of SSDL and provide an example use-case for its application to the problem of designing a smart service for parking lot management.

R. Ruiz-Torrubiano (✉) · D. Dhungana · G. Kormann-Hainzl · S. Paudel
IMC University of Applied Sciences Krems, 3500 Krems, Austria
e-mail: ruben.ruiz@fh-krems.ac.at

D. Dhungana
e-mail: deepak.dhungana@fh-krems.ac.at

G. Kormann-Hainzl
e-mail: gerhard.kormann@fh-krems.ac.at

S. Paudel
e-mail: sarita.paudel@fh-krems.ac.at

J. Meierhofer et al. (eds.), *Smart Services Summit*, Progress in IS,
https://doi.org/10.1007/978-3-031-36698-7_12

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 (Maglio et al., 2009). This configuration includes in general people, organizations, shared information and technology. Service consumers and 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 (Chandler & Lusch, 2015). 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 (Lim & Maglio, 2018). 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 city context, smart service systems enable value-cocreation for the participants of the smart city ecosystem to address its main challenges (like urbanization, climate change, sustainable transport, housing, and healthcare) by an intelligent use of information technologies (Wolff et al., 2020). One of the main challenges to deliver these goals is an efficient way of engineering smart service systems to enable the city and its citizens to harness the large volumes of data that are produced by sensors and digital infrastructure to improve sustanability by service innovation (Dobler et al., 2021; Suzic et al., 2022). In this paper, we address this software engineering challenge by proposing a low-coding approach based on a domain-specific modeling language that we call Smart Service Definition Language (SSDL). The basic goal of this language is to provide a means of defining, sharing and deploying smart service systems in a natural way also for non-technical users, enabling the relevant stakeholders in the city administration to autonomously define and deploy smart service systems based on their needs and the available infrastructure. SSDL is the foundation for a low-coding platform capable of generating applications based on a simple formal description, largely simplifying the development of smart services in practice.

This paper is organized as follows. In Sect. 2 we begin by providing a review of related work in the field of smart systems design with a focus on smart cities. Section 3 outlines the research and design methodology used and provides the syntax and semantics definitions for SSDL. In Sect. 4 we describe a general software architecture for implementing a low-coding platform for smart service systems based on SSDL. Our approach is evaluated in Sect. 5 by applying it to an example use case. We conclude in Sect. 6 with a summary.

2 Related work

Smart products integrate resources and activities of service providers and service consumers, which allows them to adapt the service systems based on contextual data (Beverungen et al., 2019a). In Beverungen et al. (2019b), the authors present

the concept of smart services and smart service systems. As smart products are widely used for smart sevices in smart cities the related work helps us better understand our target domain. Jussen et al. (2019) present a service-engineering approach for smart services and illustrates its successful application and its impact on a medium-sized company. The study focuses on the development steps involved and the interaction and interconnection of elements in smart services. Komninos et al. (2016) propose an ontology for smart cities called Smart City Ontology (SCO) and present its building blocks (e.g., technology, structure, functions, design), and properties for connecting those blocks in the ontology. This study was enhanced in Komninos et al. (2021) with an OWL ontology. In this work, we use this OWL-based Smart City Ontology to provide the language with semantic and syntactic features related to smart cities, therefore these related works help us to understand the smart city landscape, identify its main components and the processes involved in smart city applications. In Huber et al. (2019) the authors develop a domain-specific modeling language for smart service systems and demonstrate it by presenting real-world scenarios. This work focuses on concepts of domain-specific modeling and relationships in smart service system domains, highlighting the importance of developing domain-specific modeling languages for better coverage of the target domain. Similarly, guidelines for selecting an appropriate metamodeling language are presented in Frank (2013). The authors demonstrate a process for specifying a domain-specfic modeling language based on requirements analysis. We considered this work as our starting point for designing our domain-specific modeling language.

3 Smart Service Definition Language

SSDL is a text-based language. An SSDL file contains the definition of a smart service. For the general structure, we borrowed some syntactic elements from YAML,[1] as this language provides syntactic constructions that can be regarded as near to natural language. We then enriched these constructions by defining additional semantic elements from the Smart City Ontology. We followed the general methodology for designing domain-specific modeling languages proposed in Frank (2013). This methodology comprises the following steps: (1) clarification of scope and purpose, (2) analysis of generic and specific requirements, (3) language specification, (4) design of graphical notation, (5) development of a modeling tool, and (6) evaluation and refinement.

Table 1 shows a summary of the syntax of SSDL. The preamble contains metadata, i.e. general data about the smart service itself, mainly for documentation purposes. Two metadata-fields are mandatory: `name` and `version`. Additional metadata-fields can be used to define the general area that the smart service belongs to, like environment or transportation.

[1] https://yaml.org/.

Table 1 Structure of the Smart Service Definition Language (SSDL)

Section	Description	Elements	Example
Service	Metadata, documentation, classification	`name`	`name is "Waste Management"`
		`version`	`version is 1.0`
		`domain`	`domain is Infrastucture`
		`field`	`field is Environment`
Data	Configuration data sources, sensors, gateways, formats	`name`	`name is "Waste Bins"`
		`type`	`type is WasteSensor`
		`provider`	`provider is Fiware`

		`config`	`config:`
			` url is http:...`
			` token is 0309f94...`

		`query`	`query:`
			` sensor: urn:...:123`
			` property: capacityRemaining`
			` property: location`
			` property: time`

		`format`	`format:`
			` capacityRemaining is number`
			` location is geodata`
			` time is timestamp`

Application	Service definition and data visualization	`type`	`type is WebApplication`
		`layout`	`layout is SinglePage`
		`roles`	`roles:` ` administrator, user`

		`visualization`	`visualization:`
			` type is LineChart`
			` x is time`
			` y is capacityRemaining`

Deployment	Deployment profiles	`type`	`type is Docker-Compose`
		`file`	`file: docker-compose.yaml`

		`credentials`	`credentials:`
			` username: admin@smart.city`
			` password: 123adjikj!...`

The `data` section contains definitions regarding the data sources used, which typically belong to the sensor infrastructure already deployed in the smart city. In this section, one or more data sources can be defined using a YAML-like list notation. The data sources themselves can be specified using the following properties: the `name` of the data source, the `type` of the data source used and the name of the Internet-of-Things (IoT) `provider` (e.g. Fiware[2]). Compound objects are used to specify additional details: `format` specifies the data returned by the data source, including property names and types, `config` describes how to authenticate against the IoT platform, and `queries` defines how to query specific sensors or entities.

The `application` section contains the declarations needed for specifying the smart service itself. This includes properties like the type of the application and how to display data. The properties that can be specified are: the `type` of the application, which will be a web application by default, the `layout` (like single-page or other types of layouts), the standard user `roles`, like administrator and a read-only user type, and the data `visualization` used, like several types of plots or tables.

The `deployment` section is optional and specifies deployment environments where the smart service will be published and made available to the end users. A deployment environment is a runtime or framework being executed on the server that runs the smart service system. By defining the runtime environment in the SSDL file, the user can use tools for automating the deployment process and thus decrease time-to-market as much as possible.

4 Architecture

In this section, we outline a general software architecture for a low-coding platform based on SSDL. An overview of this architecture is given in Fig. 1. We begin with the user interface (UI) component at the bottom, which enables the user to interact with the system. The central element of the UI is a text editor featuring syntax highlighting (rendering language keywords and symbols differently) and validation (providing visual cues like underlining incorrect syntax and explanations in text form).

Low-coding features like pre-defined blocks of code that can be dragged directly into the editor are also integrated in the UI. For instance the user can choose from a sidebar between different pre-defined components like default data blocks (data section), default application (web applications) and deployment types (container-based deployment). The configuration of these blocks is done visually by means of configuration dialogs that can be used to save the configuration or modify it later.

The backend is organized as a service-oriented architecture (SOA) composed of several independent services, each providing a subset of the needed functionality. The main services composing the backend are the `ApplicationService`, the `LanguageService` and the `DataService`. The `ApplicationService` is the central business logic service for the web application that communicates with

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

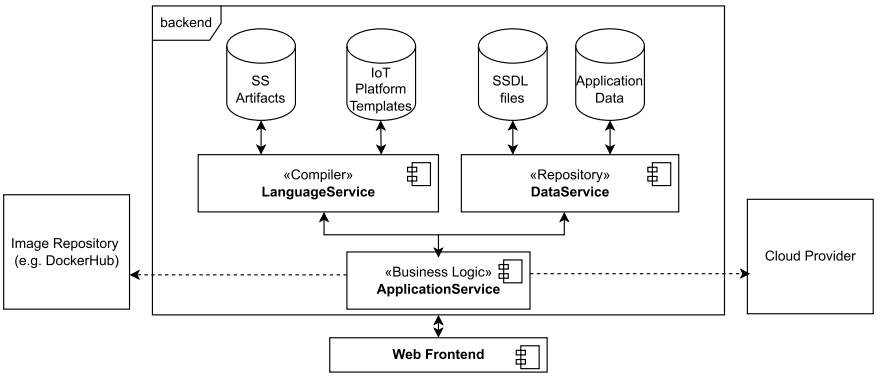

Fig. 1 An architectural overview for a low-code platform based on SSDL

the frontend to provide the requested functionality. Its main task is to serve as a controller for the features supported by the UI and a proxy to the other services by exposing the necessary APIs. The `LanguageService` implements a compiler for SSDL and generates the source code of the corresponding smart service (which we call the *artifact*). This service is also responsible for handling validation requests from the UI, which are forwarded by the `ApplicationService`. The output of the `LanguageService` is a fully fledged web application based on a general programming language (like Java or Python). The compiler generates the code of the application based on *templates*, which are in essence source files with placeholders that are filled at compile time. These templates can be either general (concerned with the general structure of the application and the main business logic) or IoT-specific, providing modules implementing API clients for the target IoT platform. For instance, if the sensor layer is based on the Fiware framework, the template that implements the Fiware API is selected by the compiler. The implementation is assembled in compile time with the smart service so that the service, when deployed, can access and fetch data from the corresponding sensor layer. This encapsulates the concrete data modeling for a specific IoT vendor and makes SSDL independent of the specific IoT platform used. Finally, the `DataService` is responsible for managing a repository of source SSDL files. These files are kept in a data store. Additionally, this services manages all the other types of data needed by the application, like user data.

5 Evaluation

We now present a use-case solved using the proposed domain-specific language in the smart city context: smart parking. The problem of managing the available parking space in modern cities has practical importance for both the drivers and the operators

and has sustanability implications. There are several specific variants of this problem depending on the goals and the technology used (Al-Turjman & Malekloo, 2019). In general, smart parking services alleviate congestion in city centres and thus reduce vacant slot search time which also results in reduced air pollution. The sensing layer consists in sensors placed in the available parking space so that the presence or absence of a vehicle can be measured. The sensors then transmit measurements using the corresponding network infrastructure to the IoT middleware. In a traditional smart service engineering approach, dedicated application development would be needed to build a tightly coupled application that communicates with the IoT middleware to fetch data and implement some functionality related to the data. In contrast to this approach, we now build an SSDL smart service definition in Fig. 2. In this definition, sensors of infrared type and four data fields of interest are specified: an identifier for the sensor making the measurement, the timestamp of the measurement, a boolean indicating if the parking slot is vacant or not, and the location of the parking slot. When compiled, this section would generate an implementation of the NGSIv2 REST API.[3] Next, the application is defined as a single-page web application with standard administrator and user roles. The application shows the data in two ways: first, it displays a map showing the city centre within a radius of 10 km where the data points are defined by their GPS coordinates and the boolean indicating if the spot is vacant or not. Additionally, it shows the data as a table. Note that all that we need is to specify a few lines of text instead of writing a full web application from scratch. This lowers the costs of developing such an application and at the same time serves as an implementation guide for further smart cities interested in introducing smart parking solutions themselves.

6 Conclusions and Future Work

In this paper, we presented a new domain-specific modeling language and a general software architecture for implementing a low-coding approach for developing smart services in the smart city context. The main contributions of the present study are the following: first, a general and flexible declarative formal language for automatically generating smart service applications was proposed. This language is designed by using a principled methodology and resembles natural language to booster acceptance by non-technical users. Second, SSDL represents the foundation for a low-coding platform for engineering smart services, allowing for implementing a wide variety of features for supporting the user (e.g. graphical modeling tools). Third, our approach is IoT platform agnostic, which means that it can work with arbitrary data models and middlewares. Adaptation to new platforms can be done by implementing an abstract layer that supports the corresponding middleware API.

Current ongoing work includes the implementation of a platform prototype to demonstrate its feasibility. Visualization features (like representing code blocks to be

[3] https://fiware-orion.readthedocs.io/en/latest/.

```
service:
  name is "smart_parking"
  version is 1.0
  domain is Infrastructure
  field is Transportation
data:
  - sensor1:
    provider is Fiware
    config:
      endpoint: http://...
      token: 0af2347ed...
    name is "parking_lot"
    type is Infrared
    queries:
      entity: urn:parking:434
      property: id
      property: time
        timestamp
      property: vacant
        bool
      property: location
        geodata
application:
    type is Application.Web
    roles:
      administrator, user
    visualization:
      - "map1":
        type is Map
        center is 48.21,16.36
        data:
          x is location
          y is vacant"
      - "table1":
        type is Table
        data:
          - id, timestamp,
          location, vacant
deployment:
  type: Docker-Compose
  file: "docker-compose.yaml"
```

Fig. 2 Example SSDL file for a smart parking service

used graphically, and using drag-and-drop for connecting these components) should be investigated to determine their viability as well. The prototype will be further developed to implement an example use case from end to end, providing a sample implementation for first production usages.

References

Al-Turjman, F., & Malekloo, A. (2019). Smart parking in IoT-enabled cities: A survey. *Sustainable Cities and Society, 49,* 101608.

Beverungen, D., Breidbach, C. F., Poeppelbuss, J., & Tuunainen, V. K. (2019a). Smart service systems: An interdisciplinary perspective. *Information Systems Journal*, *29*(6), 1201–1206. Publisher: Blackwell Publishing Ltd.

Beverungen, D., Müller, O., Matzner, M., Mendling, J., & vom Brocke, J. (2019b). Conceptualizing smart service systems. *Electronic Markets*, *29*(1), 7–18. Publisher: Springer Verlag.

Chandler, J. D., & Lusch, R. F. (2015). Service systems: A broadened framework and research agenda on value propositions, engagement, and service experience. *Journal of Service Research*, *18*(1), 6–22. Publisher: SAGE Publications Inc.

Dobler, M., Kalkhofer, H., & Schumacher, J. (2021). Smart service development in public-private settings–assessment methodology and use-cases in the lake constance region. In S. West, J. Meierhofer, & C. Ganz (Eds.), *Smart services summit* (pp. 3–13). Cham. Springer International Publishing.

Frank, U. (2013). Domain-specific modeling languages: requirements analysis and design guidelines. In *Domain engineering: product lines, conceptual models, and languages* (pp. 133–157). Journal Abbreviation: Domain Engineering: Product Lines, Conceptual Models, and Languages.

Huber, R. X. R., Püschel, L. C., & Röglinger, M. (2019). Capturing smart service systems: Development of a domain-specific modelling language. *Information Systems Journal,29*(6), 1207–1255.

Jussen, P., Kuntz, J., Senderek, R., & Moser, B. (2019). *Smart service engineering* (Vol. 83, pp. 384–388). Elsevier B.V.

Komninos, N., Bratsas, C., Kakderi, C., & Tsarchopoulos, P. (2016). Smart city ontologies: Improving the effectiveness of smart city applications. *Journal of Smart Cities*, *1*(1).

Komninos, N., Panori, A., & Kakderi, C. (2021). *The smart city ontology 2.0: Assessing the components and interdependencies of city smartness*. Publisher: Preprints.

Lim, C., & Maglio, P. P. (2018). Data-driven understanding of smart service systems through text mining. *Service Science*, *10*(2), 154–180. Publisher: INFORMS.

Maglio, P. P., Vargo, S. L., Caswell, N., & Spohrer, J. (2009). The service system is the basic abstraction of service science. *Information Systems and e-Business Management, 7*(4), 395–406.

Suzic, B., Urban, S., Hellwig, M., & Dobler, M. (2022). Smart circular economy value drivers: The role of the financial sector in stimulating smart regional innovation-driven growth. In S. West, J. Meierhofer, & U. Mangla (Eds.), *Smart services summit* (pp. 55–64). Springer International Publishing.

Wolff, A., Barker, M., Hudson, L., & Seffah, A. (2020). Supporting smart citizens: Design templates for co-designing data-intensive technologies. *Cities, 101,* 102695.