A Framework to Support the Development of Self-adaptive Service-oriented Mobile Applications

William Filisbino Passini[1] and Frank José Affonso[2]
Department of Statistics, Applied Mathematics and Computation
São Paulo State University – UNESP
PO Box 178, 13506-900, Rio Claro, SP, Brazil
[1]william.passini@gmail.com, [2]frank@rc.unesp.br

***Abstract*—Today's society is increasingly dependent on the use of mobile devices, which have changed over these last 10 years the way people perform their daily tasks. This can certainly be one of the factors that has boosted the demand for development of high-quality Mobile Applications (MobApps). In short, to improve the efficiency of the development life cycle, these applications often use third-party components (e.g., software components, web services, and other mobile applications). Service-oriented MobApps have been a feasible alternative to overcome hardware limitations of these devices to increase the processing and storage capacity. In another perspective, it is also noted a change in the behavior of users of MobApps and their needs, which require applications capable of modifying their structure and/or behavior at runtime. Thus, this paper presents a framework to support the development of Self-adaptive Services-oriented MobApps (Self-MobApps), which enable adaptation of services at runtime. To show the feasibility of our framework, a case study for a smart restaurant was conducted. The results of this study enable us to create a positive perspective on the contribution of our framework to the research communities involved.**

***Index Terms*—Framework; Mobile Applications; Service Computing.**

I. Introduction

The complexity of software systems and their computational environments has increased in the last years. Nowadays, our society has become increasingly dependent of such systems, which must be able to work in 24/7 mode (i.e., 24 hours per day, seven days per week). Thus, it can be noted that most human daily tasks are managed by applications embedded (i.e., Mobile Applications – MobApps) into mobile devices (e.g., smartphones or tablets), which allow on-line access to information regardless of the users' location [1], [2], [3]. Regarding the development, such applications can have, at the same time, some items: (i) ad-hoc components and applications developed by third-party; (ii) on-line services; and (iii) platform-dependent components to access device-specific hardware (e.g., camera, GPS – Global Positioning System, microphone, among others) [4].

Mobile devices have some physical limitations (e.g., processing and storage) compared to personal computers. For these reasons, research has been boosted to minimize the impact of such limitations and, at the same time, to facilitate access to information providing mobility to their users. Based on the presented context, the integration of MobApps into SOA-based (Service-Oriented Architecture) systems have been a feasible alternative to overcome these limitations. In short, SOA provides an architectural model that enables services to be published by service providers, discovered and consumed by the stakeholders (i.e., client applications, other services, among others) by means of platform-independent communication process based on set of XML-based (eXtensible Markup Language) standards [1], [5], [6].

SOA-based systems have played an important role in the development of distributed applications over the Internet. In this context, web services can be considered elements of first class to support the development of applications based on services in heterogeneous environments. Moreover, such applications, regardless of type (e.g., distributed, mobile, or web), must be prepared to deal with the changes at runtime, which can be performed to meet the user's new needs (e.g., new requirements) or autonomously react to modifications in their execution environment (e.g., services unavailability). Therefore, services that enable adaptation at runtime can be classified as self-adaptive service [4], [7].

Based on presented context, a framework to support the development of Self-adaptive Services-oriented MobApps (Self-MobApps) is proposed in this paper. In short, this framework enables services to be monitored by a supervisor system and adapted at runtime. Such system was designed by our research group in previous work [8] and enables to classify and analyze sensory data to autonomously detect and mitigate faults at runtime (e.g., service unavailability, failures, or high response time). Moreover, this framework aims to support the development of Self-MobApps by means of a dynamic approach for service deployment. In other words, unavailable services can be replaced by a similar one in a transparent way without the perception of their stakeholders (i.e., client applications). For reasons of scope, our framework addresses only services based on JAX-WS (Java API for XML Web Services) [9].

The paper is organized as follows: Section II presents the background and related work; Section III provides a description of our framework; Section IV presents a case study to show the applicability of our framework; and Section V summarizes our findings, conclusions, and perspectives for further research.

II. Background and Related Work

This section presents the background (i.e., concepts and definitions on self-star software and self-adaptive services) and related work that contributed to the development of our framework.

DOI reference number: 10.18293/SEKE2018-163

Self-star software. Self-adaptive Software (SaS) has specific characteristics compared to a traditional one because this type of software enables structural, behavioral, or contextual changes at runtime. Among these changes, some of them deal with management of complexity, robustness in handling unexpected conditions (e.g., failure), changing priorities and policies governing the goals, and changing conditions (e.g., execution environment). According to Salehie & Tahvildari [10], "SaS is expected to fulfill its requirements at runtime in response to changes. To achieve this goal, software should have certain characteristics, known as self-* properties (...). These properties provide some degree of variability, and consequently, help to overcome deviations from expected goals (e.g., reliability)". To manage the changes at runtime, feedback-loop proposed by IBM [11] has been a good alternative, since all decisions are taken based on a plan established in the data collected from execution environment.

Self-adaptive service. Li et al. [12] proposed a self-healing framework for QoS-aware (Quality of Service) web services composition. Self-healing is a self-property that provides special ability to software systems, which can perceive that they are not operating correctly and, without human intervention, make the necessary adjustments to restore them to normal operation. To do so, this framework uses Case-Based Reasoning (CBR) for using previous experiences to understand and solve new problems by means of service similarity. A knowledge-based approach for Service Composition based on self-healing was developed by Angarita et al. [5]. SC is an application composed of a service set that interacts with each other and is invoked on the Web. This type of service can be classified in two categories: (i) static, which represents the aggregation of services taken place at design time; and (ii) dynamic, which enables determining and replacing services at runtime [13].

As related work, Sefid-Dashti & Habibi [14] proposed a mobile SOA Reference Architecture (RA) based on 26 mobile SOA patterns and introduced a new domain specific layer. This RA enables reaction to changes in infrastructure in order to streamline a service, which can be realized by several mobile SOA patterns. A SOA-based platform-specific framework for context-aware MobApps was developed by Daniele et al. [15]. This framework was based on a RA composed of components typically used by MobApps by means of automated design approach. Moreover, the design of context-aware MobApps is platform-independent and can be realized with different specific target implementations. Cherif et al. [6] proposed a Reference Model for specifying Self-adaptive Service-based Applications (ReMoSSA). This model was based on FORMS model [16] and the automate element proposed by IBM [11]. Moreover, costs and efforts of maintenance can be minimized, since this model enables to inspect if the dynamic monitoring and the dynamic adaptation are being considered in the design phase. A declarative approach called SelfMotion (Self-Adaptive Mobile Application) was designed by Cugola et al. [4]. In short, applications based on SelfMotion can be performed by a middleware that enables to create at runtime the best sequence of abstract actions (i.e., service orchestration) to achieve the goals, mapping them to the concrete actions to execute in accordance with the specified QoS Policy. Such sequences are elaborated by automatic planning techniques, which enable the service changes without perception of their stakeholders. Finally, Nasridinov & Byun [17] proposed a framework named WS–DIRECT (Web Service–DIscoverability, REcoverability, Classifiability and Trustworthiness). This framework provides a set of mechanisms to deal with service adaptations at runtime, such as: (i) semantic discovery; (ii) self-healing, i.e., monitoring, diagnosis and repair; (iii) classification of QoS; and (iv) ontology-based security. Li et al. [12] designed a framework QoS-aware web service composition based on self-healing property and CBR. This framework enables to propose solutions for detected problems, to make assumptions and predictions based on previous experiences, and to adapt to changes of the environment.

III. Framework for Self-MobApps

According to Erl [18] and OASIS [19], SOA is composed of three elements: (i) web service provider, which is responsible for providing the services that will be executed; (ii) web service repository, which is responsible for describing, publishing, and finding services; and (iii) web service client, which represents the consumers of such services. Regarding the second element, it represents an XML-based standard called UDDI (Universal Description, Discovery, and Integration). In short, it enables the registry of all web service's metadata, including a pointer to the WSDL (Web Service Description Language) description of a service, beyond a set of WSDL port type definitions for manipulating and searching such registry. Figure 1 shows the general representation of SOA.

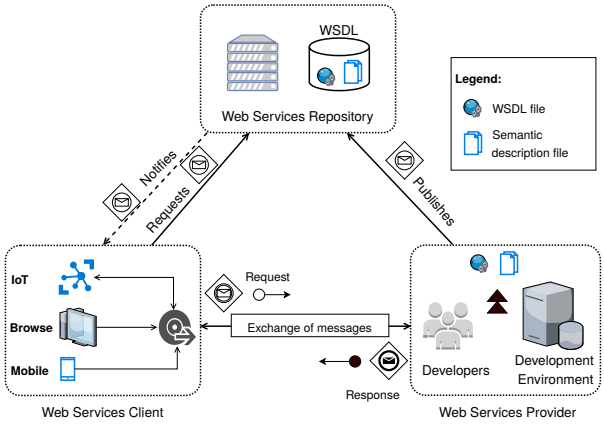

Figure 1. Service-oriented Architecture (Adapted from [19])

Based on these concepts, this section presents a framework to support the development of Self-MobApps. This framework enables services to be monitored at runtime and replaced in case of problems (e.g., service unavailability, failures, or high response time). With regard to the replacement of services, two possibilities are allowed by our framework: design time and runtime. The first uses a list of preferred services defined by

the developers in the design phase of a primary service. The second uses an automatic mechanism of search to find a similar service in the web service repository. Finally, our framework addresses only services based on JAX-WS and simplifies creating and deploying web services and web services clients. Figure 2 shows the general representation of our framework, which is composed of a core for adaptation (dotted line) and four additional modules: development, search, action plan, and deploy. Next, a brief description of this architecture is addressed.

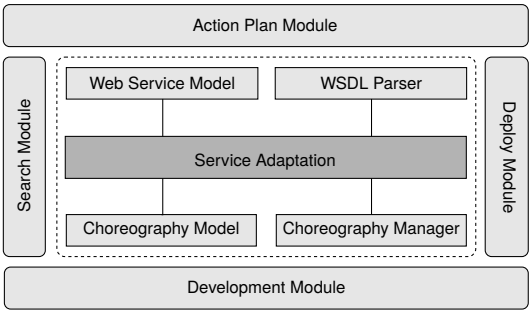

Figure 2. Framework for Self-MobApps

Development module. This module provides a set of guidelines for Self-MobApps development. Web services are designed by the developers and registered into environment execution (i.e., Web Service Repository – Figure 1). These services are composed of WSDL and semantic description files. The first one provides to our framework technical description about operations embedded into a service that will be used by our "WSDL Parser" for parameter matching. The second one contains textual description of a service that will be used by "Search Module" when a new service is requested. Moreover, it is noteworthy that when a composition of services (i.e., primary and preferred services) is developed, the software engineer can associate a list of alternative services to each preferred service of this composition. This list must be used when the primary service presents execution problems (e.g., unavailability, poor performance, among others) and a preferential service can assume its role.

Search module. This module aims to assist in the search of services in the repository when an adaptation activity is invoked (Figure 1). To do so, three search methods are provided: (i) semantic, which can be defined as a search query by means of contextual meaning for services. This type of search provides more meaningful results by finding the most relevant service in the repository; (ii) technical, which can be specified only with service information in template format, which is converted as input parameters to search in the service repository for matching operation; and (iii) quality, which can be defined as the description or measurement of the overall performance of a service. According to [20], unresolved QoS issues may cause serious problems in relation to execution (e.g., unacceptable levels of performance degradation). Thus, our framework addresses seven quality attributes [20], [21]: availability, accessibility, integrity, performance, reliability, regulatory, and security. These attributes represent minimal quality of a service. However, other attributes can be found in the literature.

Action plan module. This module aims at assisting in the adaptation activity of services providing means to control dynamic behavior, individual reasons, and execution state of each service in relation to the execution environment. To do so, a framework for decision-making in SaS developed by Affonso et al. [8] in previous work was used. In short, such framework is composed of two modules: classification and recommendation. The main purpose of the first module is to present a classification for a data set collected via sensors from execution environment. The second module aims to present a solution set ranked by statistical measures for a problem reported by the classification module.

Deploy module. This module aims to support the deployment process for Self-MobApps. In other words, a service can be deployed, undeployed, and redeployed. To do so, we have used jUDDI implementation [22], which is an open source Java implementation of OASIS's UDDI specification for Web Services. When a service is inserted into repository (Figure 1), its WDSL file is parsed into a structure that aims to provide information about services and the operation of such services. Then, for each operation, its parameters and types are retrieved. In addition, technical service information (e.g., namespace and URL (Uniform Resource Locator) service) must also be obtained.

Core of adaptation. This structure can be considered the "heart" of our framework, since enables managing service adaptation at runtime. Basically, this core is organized in five modules: (i) Web Service Model, (ii) WSDL Parser, (iii) Choreography Model, (iv) Choreography Manager, and (v) Service Adaptation. Figure 3 shows the UML model for core of service adaptation. Next, a brief description of each module is reported.

Web Service Model. This module contains the model elaborated for the representation of a web service (`parserManager.model` package – Figure 3). From this point onwards, this model may be also referred to as WSModel (Web Service Model). The `WebService` class is composed of five attributes that describes a service. Each web method of a service has none or many parameters (`Parameter` class).

WSDL Parser. This module (`parserManager.parser` package – Figure 3) contains a set of classes responsible for transforming a WSDL document (i.e., file or URL) into WSModel (`parserManager.model` package). Thus, from such document, relevant information of a service is extracted by means of DOM (Document Object Model) API. The main purpose of this operation is to read an XML file and parse its content into a tree structure, where each node is composed of XML document tags.

Choreography Model. This module has the model elaborated for the representation of a choreography that must be executed in the service (`wsManager.model` package – Figure 3). The `WebService` class represents the list of web methods that can be executed by a service and the respective parameters

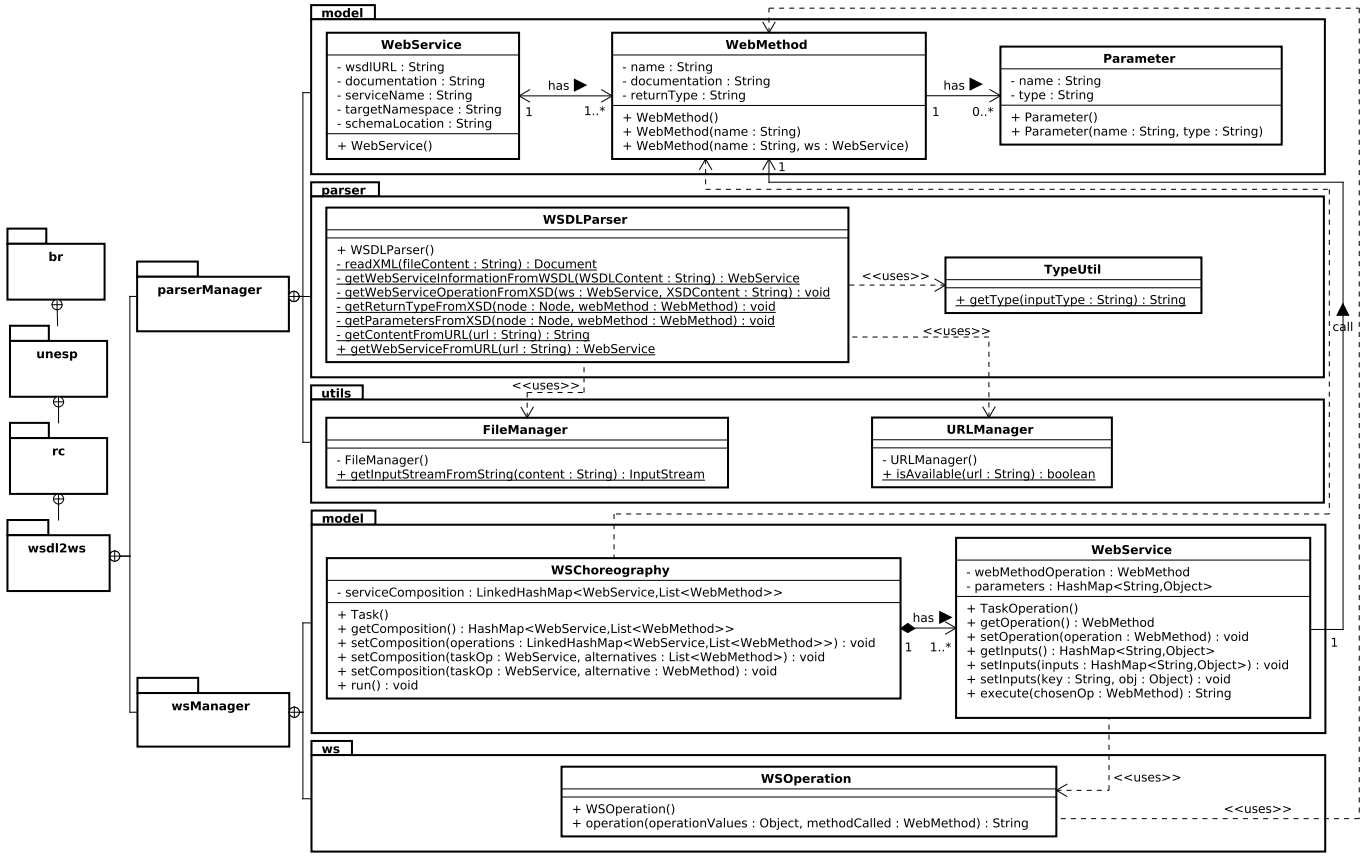

Figure 3. UML model for core of service adaptation

that must be provided to each method at runtime.

Choreography Manager. This module has only the `WSOperation` class (`wsManager.ws` package – Figure 3), which is responsible for acting as a client application in the communication process via web service. In short, this class has only the `operation` method that enables executing a web method of a service via parameters.

Service Adaptation. This module can be considered as RA4Self-MobApps "orchestrator", since it performs calls and coordinates all activities of the other modules (i.e., Core of adaptation). In short, this module can be defined as a supervisor system of web services, monitoring their requisitions in the execution environment. To do so, this module implements a well-defined process to adapt a web service at runtime. Finally, this process must be performed automatically by software engineering tools, intending to reduce implementation complexity and minimize uncertainties generation.

IV. Case Study

To evaluate the applicability, strengths, and weaknesses of our framework this section presents a case study we have conducted. As subject application of our empirical analysis, we have selected an application addressed to the management of a smart restaurant called App2Rest. This restaurant provides a table set for its customers, which are equipped with a device set that aims to automate the restaurant service from orders to payments. Next, a brief description of our subject application and the empirical strategies adopted for conducting this case study is presented.

Subject Application. The App2Rest was organized in two layers: (i) Devices, which represent the means of access to the clients (i.e., front-end side) of this application, which can use mobile devices equipped with the Android operating system (i.e., smartphones and tablets) and personal computers via the Web; and (ii) Web Server, which represents an application composed of a service set (i.e., back-end side). Regarding the exchange of information between these two layers, a subsystem was developed to serialize application models via JSON (JavaScript Object Notation) and facilitate the exchange of information between these layers (i.e., client and server). For instance, when a menu is requested by the client side, a query is performed on the server side and a model composed of several classes (i.e., data) is instantiated. Next, such model is serialized to a String (i.e., JSON format) and transferred to the client side. The inverse process is also considered.

Empirical research strategy. Figure 4 illustrates the generic structure of our application, which is organized in a client-server architecture composed of two layers. Regarding the operation of this application, all requests of the "Devices" layer (i.e., client side) are sent to the "Web Server" layer, whose purpose is to intermediate the communication (i.e., message exchange) between customer or restaurant devices

and requested web services of such application. The web services contained in the App2Rest can be classified in two levels of complexity: (i) simple service, which represents the encapsulation of a functionality to be made available to its customers (i.e., "Devices" layer); and (ii) service composed of services, which represents the encapsulation of more than one functionality of the restaurant application to be made available to its customers. This service type executes a call set to other services by means of a choreography (i.e., action sequences and conditions) for its functionality to be fulfilled. For instance, the authentication of a customer in the restaurant application is a simple service. Customers request orders (i.e., items for an order) can be characterized as a composite service.

Regardless of the complexity level, the App2Rest has a "Service Monitor" component in each service to monitor its execution status (Figure 4). In a first analysis, this monitor will determine if the web service that is being requested is available to be accessed by a customer application (i.e., "Devices" layer). In a more refined analysis, the quality of such services can also be assessed. In both analyzes, one service can be replaced by another equivalent at runtime without client's perception. In this sense, to present the details of this last phase, the `WS_01` service will be considered, which enables customers to make orders for the restaurant. This service is designed based on three services (`WS_A`, `WS_B`, and `WS_C`) by means of a service composition. The first service (`WS_A`) aims to authenticate the customer in restaurant application via "`E-MAIL`" or "`GOOGLE SIGN IN`". Once authenticated, the App2Rest displays the main screen and the meal of the day is suggested. Next, the customer can accept this suggestion or select another dish via left side menu. To select another dish, the customer must access the `Menu` option in the side menu so that the dishes and all items available in the restaurant are displayed. The restaurant menu is performed by the second service (`WS_B`), which enables select some items by category. When selecting a dish (add button), an order pad is initialized and many items can be added to it (third service – `WS_C`). Next, a brief description of each phase is addressed.

In the "design" phase, developers must select the preferred services that will compose the primary service (e.g., `WS_01`). These services are inserted into a dynamic list called "Preferred Services" (i.e., `WS_A`, `WS_B` and `WS_C`, and the position of each service in such list is equivalent to its execution order. More complex choreographies require an external file to indicate the order of execution of each service. In addition, for each preferred service there will be a list of "Alternative Services". For instance, for the preferred service `WS_A`, the alternative services `WS_A1`, `WS_A2` and `WS_A3` were selected.

In the runtime phase, primary services are monitored by the "Service Monitor" component. For instance, when the monitoring activity detects problems in one of the preferred services that compose a primary service, the list of alternative services is consulted. Thus, three situations can be observed: (i) the list has an alternate service that can replace a preferred one; (ii) the alternative service list may be empty, i.e., the developer has not prepared alternative services at the design phase; or (iii) the list may be empty by attempts, i.e., services have become unavailable over time. In the last two situations, the "Service Monitor" component should launch a search in the "Web Service Repository" to find a service of greater similarity. Such search can return a service set ranked by statistical measures in relation to the satisfaction of the parameters initially provided.

V. Conclusions and Future Work

This paper presented a framework that intends to support the Self-MobApps development. Such framework uses a dynamic approach for service deployment. In other words, unavailable services can be replaced by a similar one in a transparent way without the perception of their stakeholders (i.e., client applications). Moreover, our framework can classify and analyze sensory data to autonomously detect and mitigate faults at runtime for a simple service or a composition of services [8]. As reported in Section I, our framework will address only services based on JAX-WS [9]. Based on this scenario, the main contributions of this article are: (i) for the SaS area by providing a means (i.e., framework) that facilitates the development of Self-MobApps; (ii) for the Service Computing area, since we have proposed a framework that enables the development of Self-MobApps or SOA-based systems by means of a dynamic approach for service deployment; and (iii) for both areas, since we have developed a means to replace unavailable services at runtime without the perception of their stakeholders. In this sense, the proposed mechanism based on dynamic deploy for substitution of services (i.e., preferred and alternative) must be highlighted. The previous selection of services can further minimize the possible impacts (i.e., computational cost and time) caused by the deploy of services.

Regarding future work, at least two activities are intended: (i) conduction of more case studies or proof of concepts intending to completely evaluate our framework, including different software domains; and (ii) use of this framework in the industry, since it is intended to evaluate its behavior when it is applied in larger real environment of development and execution. Therefore, based on the content exposed in this paper, a positive research scenario can be idealized, intending to have this framework become an effective contribution to the software engineering and SaS communities.

Acknowledgment

This research is supported by PROPe/UNESP and Brazilian funding agencies (CNPq and CAPES).

References

[1] J. Aghav and N. Sharma, "A software architecture for provisioning of mobile services: An osgi implementation," in *MEMSTECH '11*. Polyana, Ukraine: IEEE, May 2011, pp. 24–27.

[2] F. Gonçalves, C. E. T. Oliveira, I. Silva, and L. Moura, "An architectural model for applications based on mobile services," in *ICCGI '07*. Guadeloupe City, Guadeloupe: IEEE, March 2007, pp. 1–6.

[3] F. Gonçalves, C. E. T. Oliveira, I. Silva, L. Moura, and F. Franca, "A software architecture for the provisioning of mobile services in peer-to-peer environments," in *ICIW '07*. Morne, Mauritius: IEEE, May 2007, pp. 1–6.

[4] G. Cugola, C. Ghezzi, L. S. Pinto, and G. Tamburrelli, "Selfmotion: A declarative approach for adaptive service-oriented mobile applications," *Journal of Systems and Software*, vol. 92, pp. 32 – 44, 2014.

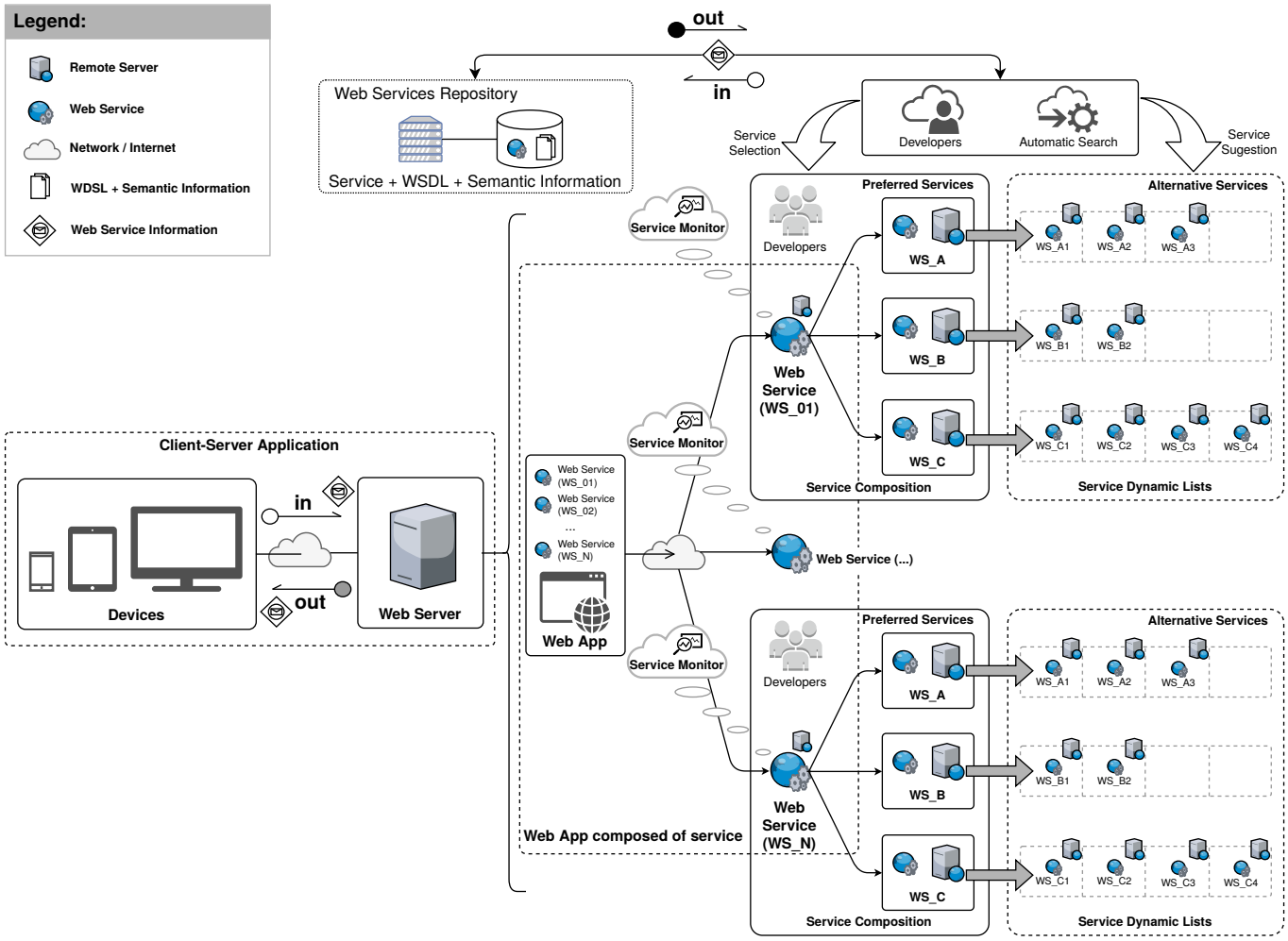

Figure 4. Generic Representation of Service for Restaurant Application

[5] R. Angarita, M. Rukoz, M. Manouvrier, and Y. Cardinale, "A knowledge-based approach for self-healing service-oriented applications," in *MEDES '16*, ser. MEDES. Biarritz, France: ACM, 2016, pp. 1–8.

[6] S. Cherif, R. Ben Djemaa, and I. Amous, "Remossa: Reference model for specification of self-adaptive service-oriented-architecture," in *AISC '14*, B. Catania, T. Cerquitelli, S. Chiusano, G. Guerrini, M. Kämpf, A. Kemper, B. Novikov, T. Palpanas, J. Pokorný, and A. Vakali, Eds. Cham: Springer International Publishing, 2014, pp. 121–128.

[7] D. Menasce, H. Gomaa, s. Malek, and J. Sousa, "Sassy: A framework for self-architecting service-oriented systems," *IEEE Software*, vol. 28, no. 6, pp. 78–85, Nov 2011.

[8] F. J. Affonso, G. Leite, R. A. P. Oliveira, and E. Y. Nakagawa, "A framework based on learning techniques for decision-making in self-adaptive software," in *SEKE '15*. Pittsburgh, USA: Knowledge Systems Institute, 2015, pp. 24–29.

[9] Oracle, "Java platform, enterprise edition: The java ee tutorial," [On-line], 2018, available: https://goo.gl/qGeGwd, Accessed on February 28, 2018.

[10] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, pp. 1–42, 2009.

[11] IBM, "An architectural blueprint for autonomic computing," [On-line], 2005, available: https://goo.gl/wawGvi, Third Edition, Accessed on February 28, 2018.

[12] G. Li, L. Liao, D. Song, J. Wang, F. Sun, and G. Liang, "A self-healing framework for qos-aware web service composition via case-based reasoning," in *APWeb '13*, Y. Ishikawa, J. Li, W. Wang, R. Zhang, and W. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 654–661.

[13] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Information Sciences*, vol. 280, pp. 218 – 238, 2014.

[14] B. Sefid-Dashti and J. Habibi, "A reference architecture for mobile soa," *Systems Engineering*, vol. 17, no. 4, pp. 407–425, 2014.

[15] L. M. Daniele, E. Silva, L. F. Pires, and M. van Sinderen, "A soa-based platform-specific framework for context-aware mobile applications," in *IFIP/IWEI '09*, R. Poler, M. van Sinderen, and R. Sanchis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 25–37.

[16] D. Weyns, S. Malek, and J. Andersson, "Forms: a formal reference model for self-adaptation," in *ICAC '10*. New York, NY, USA: ACM, 2010, pp. 205–214.

[17] A. Nasridinov and J. Byun, "Ws-direct: Web service—discoverability, recoverability, classifiability and trustworthiness," in *CUTE '12*, Y.-H. Han, D.-S. Park, W. Jia, and S.-S. Yeo, Eds. Dordrecht: Springer Netherlands, 2013, pp. 879–887.

[18] T. Erl, *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2016.

[19] OASIS, "Reference architecture foundation for service oriented architecture version 1.0," OASIS Committee Specification 01, [On-line], 2012, available: https://goo.gl/m2pEm4, Accessed on February 28, 2018.

[20] A. Mani and A. Nagarajan, "Understanding quality of service for web services, improving the performance of your web services," [On-line], 2002, available: https://goo.gl/TqkVtX, Published on January 01, 2002, Accessed on February 28, 2018.

[21] A. Al-Moayed and B. Hollunder, "Quality of service attributes in web services," in *ICSEA '10*, August 2010, pp. 367–372.

[22] jUDDI, "An open source implementation of oasis's uddi v3 specification," [On-line], 2018, available: https://juddi.apache.org/, Accessed on February 28, 2018.