# Automatic Service Composition with Heterogeneous Service-Oriented Architectures

José M. Reyes Álamo, Hen-I Yang, Johnny Wong, and Carl K. Chang

Department of Computer Science, Iowa State University
Ames, IA, USA
{jmreyes,hiyang,wong,chang}@cs.iastate.edu

**Abstract.** Service-Oriented Architecture is widely used to program pervasive spaces such as Smart Homes because of its capabilities to handle dynamic and heterogeneous environments. It is often the case that the services required are designed and implemented using different SOAs, such as OSGi and Web Services. Most of the current composition frameworks take a two-tier approach: those services following the same SOA can take advantage of service composition and runtime substitutions, while interactions between services of different SOAs require hardcoded service invocations that do not really provide full advantage of SOA. Some SOAs do not support features such as on-the-fly compositions or a searchable service directory. In this paper, we present a framework to compose and orchestrate services from different SOA implementations and provide the missing functionalities to support composition of heterogeneous SOAs. We present a case study and the performance analysis of the study to demonstrate the feasibility of our framework.

**Keywords:** Web Services, Service Composition, OSGi, SOA, BPEL.

## 1 Introduction

A service is defined as an autonomous, loosely coupled, platform independent entity that can be described, published, discovered, and invoked. It can be an atomic service performing a simple computation or a set of services that when combined perform a complex computation [1]. Service-Oriented Computing (SOC) is a programming paradigm where applications are created by having different services interacting in a logical way. Instead of depending on a single component as in the case of stand-alone applications, several orchestrated services in SOC provide the desired functionality. A fundamental characteristic of SOC is the separation of the service implementation from the service specification. Several Service-Oriented Architectures (SOAs) have been proposed for this purpose, which define the specification of services and the protocol to interact with the actual services. SOAs are designed to be loosely coupled and theoretically, any service should be able to collaborate with other services via compatible interfaces. However, in practice it is difficult for services implemented under different SOAs to interact.

Most of the research in SOA has focused on Web Services (WS). As a result, technologies and protocols for WS are well defined and studied. There exists other SOAs whose properties make them more suitable for certain applications such as OSGi, JINI, and UPnP. To comprise the full potential of SOC, we would like to be able to create composite services that are not dependent on a particular SOA. We believe that an SOA standard should be able to accommodate and utilize different services, regardless of the particular languages, platforms, and architectures they are based on. We conjecture that doing so will improve the availability and diversity of the potential applications. This will also increase the number of services to choose from as well as the quality of the composite service. Universal composition of heterogeneous services can also influences performance as we can choose the services that perform better not limiting ourselves to a particular SOA. It can also improve reliability, as a set of services might provide the same functionality but are implemented using different SOAs. If one of the services fails, the framework can select another and still be able to execute our application.

We aim at providing an efficient solution especially useful for Smart Home environments that considers the uses and properties of services. Having an approach to compose, add, remove, start, and stop these services as needed and selecting the SOA that maximizes performance is a primary motivation for this work. A scenario suitable for this kind of applications is the Smart Home environment. A Smart Home is a residential establishment equipped with sensors, actuators, and other technology to help the resident performing activities of daily living. Smart Homes are especially useful for the elderly and persons with special needs. Over the years, researchers have studied a vast number of devices and development approaches for Smart Homes. A service-oriented approach as a solution for the challenges of developing Smart Homes that satisfies the user requirements seems the most plausible. Therefore, we would like to improve Smart Home applications and move a step closer to a comprehensive solution.

In this paper, we explore the different strategies used by the research community to compose services of different SOAs. We present a composition framework that relies on services from different SOAs. We provide a working example of an actual application that combines heterogeneous services, and contrast our framework with other people's work. The rest of the paper is organized as follows: Section 2 presents related work; Section 3 discusses the importance of heterogeneous SOAs composite services; Section 4 outlines the composition framework; Section 5 describes our case study; Section 6 details the performance evaluation; Section 7 is the conclusions and future work.

## 2 Related Work

Several research works have presented strategies for automatically composing OSGi services. In [2] the authors present a framework to achieve spontaneous compositions of OSGi services. It provides functionality to handle availability of services, links to connect to the services and matching criteria for selecting the best available service. In [3] the authors use the Business Processing Execution Language (BPEL), which is a WS language, to describe the workflow of a composition. They provide a framework that extends OSGi by allowing it to interpret BPEL files, locate the services, and create the composite service. Their framework focuses on composing OSGi services only. The authors in [4] present a strategy for automatically composing OSGi services by first converting them to WS and then use WS composition techniques. Our work is different, in that we want to use different services in their original architecture without the need of converting them.

Other researchers have focused on the problem of composing heterogeneous SOAs such as OSGi services and WS. In [5] the authors support BPEL for workflow description and allow composite services to use OSGi services and WS. In their work, they add a set of custom tags to the BPEL file. These tags specify service type and binding information. These approaches are different from ours as we automatically gather this information from a service repository without extra tags. Therefore, the BPEL file needed in our framework is simpler, less verbose and represents a more interoperable solution. As there is no need to provide details about a service in the workflow specification file, this allows our framework to be extendable to other SOAs. A similar approach is followed by technologies such as the Service Composite Architecture (SCA) [5] found in frameworks such as Apache Tuscany [6], Fabric3 [7] and the Newton Framework [8]. The basic unit in SCA is a component, which is a service developed using any accepted SOA such as OSGi, WS, and Spring. SCA offers a framework where developers can create composites using services with heterogeneous implementations. They rely on annotation and a special XML language to define the components and composites. The problem with SCA is that the tags and annotations have to be included in the source code of the service so the framework can understand it. This limits the services that can be used to only those developed under a SCA infrastructure. In our work, we use the services in their original implementation, allowing us to reuse services already available.

## 3 Heterogeneous Service Composition

There is still a great need for a SOA-independent solution for service composition. The foundation of the SOA is the separation of the specification of a service from its implementation. Platform and language independence is one of the main features of SOA. However, in real life implementations it is difficult to integrate services of different SOAs. As described in the related work, some researchers have focused on automatically composing OSGi services while others attempted to combine technologies like OSGi and WS. Our work moves one-step closer towards the goal of having a platform, language, and SOA independent solution.

We present a framework for creating composite services that is extendable to accept services in different SOAs. Our solution consists of a framework capable of automatically composing OSGi services extended to include WS. In order to combine both types of services we carefully investigate different SOAs, especially WS compositions. We evaluate and integrate several existing WS protocols and technologies to accommodate for a heterogeneous set of services from different SOAs. We also examine the OSGi framework [9] that has been the subject of research for many years, it is standardized, and it has several open-source and commercial implementations available. We notice that certain very desirable features for smart home environments of the

OSGi framework are not available for WS. These features include security, dynamic installation and removal of bundles, automatic package resolution, as well as a set of essential services provided by the standard like the HTTP server and log service.

During the design and development of our framework for automatic composition of heterogeneous services we have addressed several technical challenges. One of the challenges is to ensure full support of each of the SOAs to allow seamless invocation of services. Another challenge is regarding to the lack of support for automatically creating composite services in some SOAs. We create the necessary tools to support these missing but necessary functionalities. The end-result is a composition framework that supports seamless integration of heterogeneous SOAs, both OSGi services and WS. This allows transparent heterogeneous service composition using OSGi bundles or WS without the need to modify them, as contrasts to previous approaches that rely on tags, annotations, or transformations of services.

## 4 Composition Framework

We describe how our composition framework operates in the following: First, we provide a BPEL file with the description of the composite service workflow because it is based on XML, is widely supported in the industry and the research community as well as it has become the de-facto standard for WS compositions. After invoking the composition framework with a BPEL file as input, the framework performs a syntax check. If the check finds no errors, the framework parses the BPEL file and extracts the composite service information. Using the information, it locates the candidate services in a service repository that stores information for both WS and OSGi. Once the framework gathers the services' binding information it proceeds with the actual

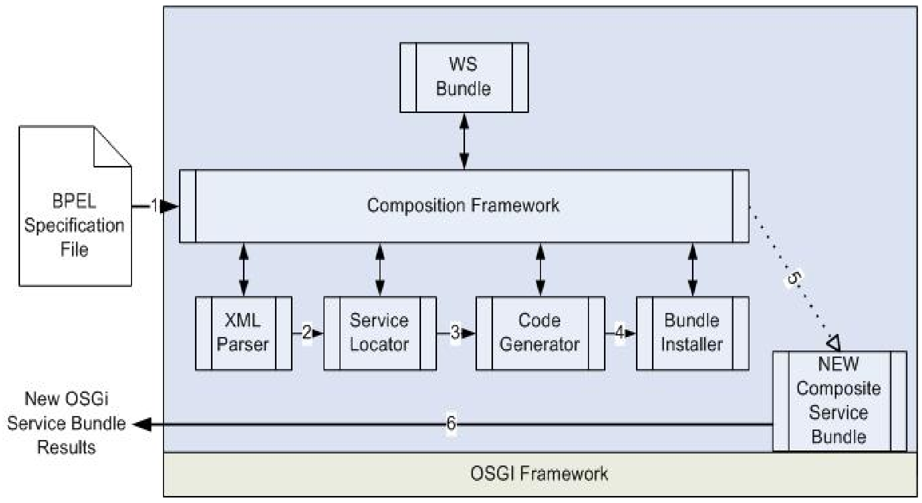

**Fig. 1.** BPEL2OSGi Composition Framework Architecture

creation of the composite service. After that, it translates the orchestration logic into executable Java program. Since the implementation of the composed service is an OSGi bundle, the framework creates the remaining components such as the service interface, the activator class, and the manifest file and includes the service references. These files are then automatically compiled, packaged into a jar file, and installed into the OSGi framework. If the service is error free it will run after a few instants otherwise, it will throw exceptions and display error messages. All these steps are automatic and transparent to the user. Figure 1 depicts a graphical representation of this framework**.** Because our composition framework takes as input a BPEL file and produces an OSGi bundle, we call our framework BPEL2OSGi.

## 5 Case Study (MISS)

We demonstrate the feasibility of our composition framework with a case study where we provide a BPEL file, create the composite service, install, and execute it. The case study is an implementation of the Medicine Information Support System (MISS) [10] that helps patients with the management of their prescriptions. It integrates the doctor, the pharmacy, and the smart home, allowing subsystems to share information with each other and uses a trusted third party to check for conflicts. Its main purpose is to increase safety by checking for conflicts among new prescription and previous medicines, health conditions and foods. By checking new prescription data with the patient's local record at each subsystem, MISS detects all possible paths of conflicts.

The workflow of MISS can be described as follows: at the doctor's subsystem, the doctor enters the new prescription's data and checks for conflicts with previous prescriptions using a trusted third party [11]. If no conflict found, MISS uses a secure communication channel to forward the prescription data to the pharmacy. At the pharmacy subsystem, MISS checks for conflicts using the patient's local pharmacy record. If no conflicts found, MISS uses a secure communication channel to forward the prescription data to the Smart Home subsystem. At the Smart Home, MISS checks for conflicts using the patient's local Smart Home record of medications, foods, and conditions.

Figure 2 shows a BPEL diagram of the workflow of MISS. This figure shows specifically the interaction between the pharmacy and the Smart Home subsystem. The process execution starts at the main node followed by the Receive Input operation, which is the default BPEL operation indicating the beginning of the execution. After that, we call a series of services. First, the Invoke RFID Service for reading the RFID tags of the medicine used by the Invoke Pharmacy Server to query the prescription information. Then we assign data to several variables and call the Invoke MCD Service [11] that checks for conflicts among the new prescription and with previous medicines and food items at home. Depending whether there is a conflict we invoke the Invoke Speech Server and Invoke Notification Service with the corresponding message. If no conflict is found the new prescription data is stored into the Smart Home subsystem by the Invoke Medicine SH Server service.

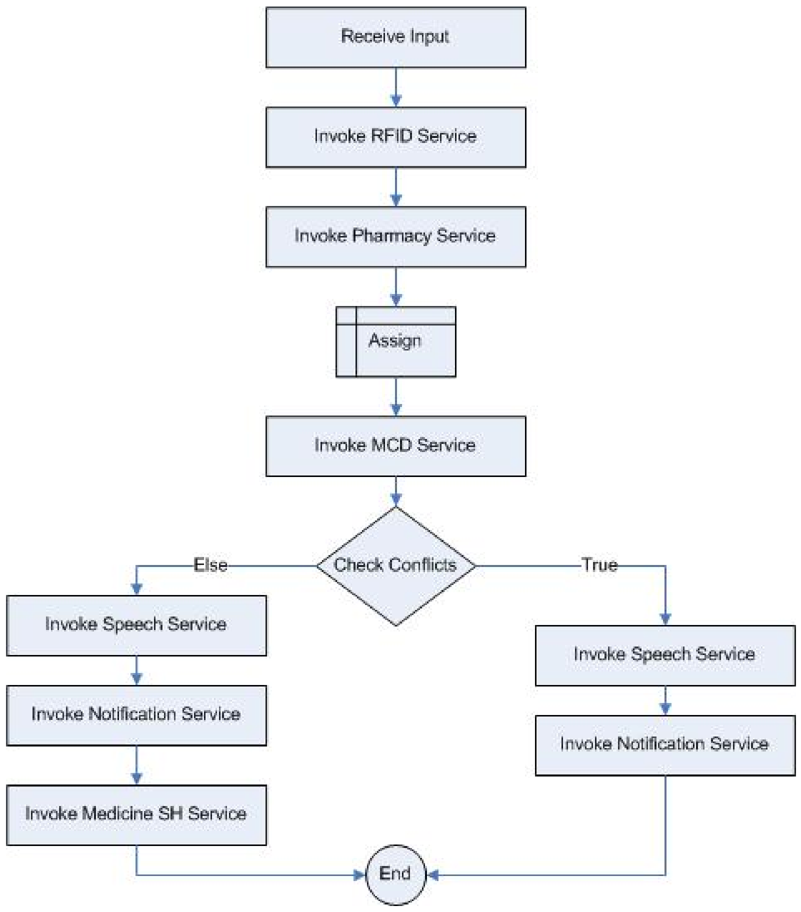

**Fig. 2.** BPEL Diagram of MISS

## 6 Evaluation

We study the performance of our composition framework by measuring the time it takes to create a composite service, how much overhead it adds and compare it to current compositions approaches. We compare the execution time of composite services using a single SOA against those using heterogeneous SOAs. In our experiments, we create a composite service that only uses OSGi services (MISS_OSGI), another that only uses WS (MISS_WS), and a third that uses a combination of both (MISS_BPEL). The overhead for parsing the XML file, locating the services, and creating the composite service is about the same in all cases. Our hypothesis is that MISS_OSGi will perform better, followed by MISS_BPEL and MISS_WS will have the worst performance. We base our hypothesis in the fact that OSGi services are local and their communication protocol is faster, while WS use HTTP that is a slower protocol. We observe a significant difference in the execution time of the composite service. Figure 3 shows the result of our experiments after we execute the three types of composite services at least 100 times and we measure the execution time.

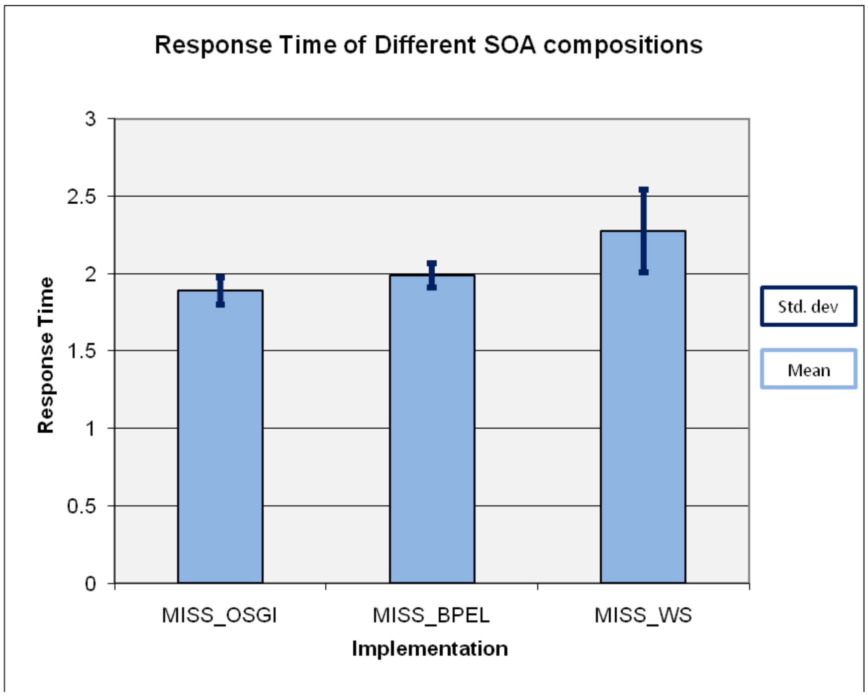

**Fig. 3.** Execution time of each implementation. The black line (left) indicates the minimum and maximum. The red line (right) indicates the standard deviation.

Our result supports our hypothesis as the MISS_OSGi had the best performance, closely followed by MISS_BPEL and the composition with the worst execution time is MISS_WS. Nevertheless, the time difference among them is in the worst case 0.3848 or less than half a second. This amount of time for this particular application is acceptable and observed by a human being, it is almost unnoticeable. The fact that WS are the slower but are the most popular and widely used, gives us an upper bound on the time that is acceptable for these applications. Another positive result is that we reduce the execution time of the worst case by using heterogeneous services. It is a great advantage the fact that we can choose different implementations of services providing the same functionality. Figure 3 also shows that the MISS_BPEL has the smallest standard deviation (and variance) of all three, making the execution and communication time more consistent and predictable.

## 7 Conclusions and Future Work

Service-Oriented Computing has been widely used in pervasive spaces such as Smart Homes. However, we would like to compose services from different SOAs. We present a framework that actually makes the automatic composition of heterogeneous services possible, requiring minimum intervention from the user. The framework provides automatic, on-the-fly compositions, a searchable service directory, and supports different SOAs like OSGi and WS.

We present a case study in which we compare the performance of the composite service created by the framework with composite services created manually. The analysis shows that the performance is acceptable sitting between the best case and the worst-case scenarios with the advantages that the framework offers.

For future work, we plan to extend this framework by adding support for other SOAs to compose a greater variety of services and study their performance. We also want to have a standardized service directory for better searching and binding to services.

## References

1. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. Computer 40, 38–45 (2007)
2. Wood, A., Stankovic, J., Virone, G., Selavo, L., He, Z., Cao, Q., Doan, T., Wu, Y., Fang, L., Stoleru, R.: Context-aware wireless sensor networks for assisted living and residential monitoring. IEEE Network 22, 26–33 (2008)
3. Redondo, R., Vilas, A., Cabrer, M., Arias, J., Lopez, M.: Enhancing Residential Gateways: OSGi Services Composition. In: International Conference on Consumer Electronics, ICCE 2007, Digest of Technical Papers, pp. 1–2 (2007)
4. Anke, J., Sell, C.: Seamless Integration of Distributed OSGi Bundles into Enterprise Processes using BPEL, Bern, Schweiz
5. Lee, C., Ko, S., Kim, E., Lee, W.: Enriching OSGi Service Composition with Web Services. IEICE Transactions 92-D, 1177–1180 (2009)
6. Apache Tuscany, http://tuscany.apache.org/ (March 23, 2010)
7. Fabric3 - Open Source SCA, http://www.fabric3.org/ (March 23, 2010)
8. Newton Framework, http://newton.codecauldron.org/site/index.html (March 23, 2010)
9. Marples, D., Kriens, P.: The Open Services Gateway Initiative: an introductory overview. IEEE Communications Magazine 39, 110–114 (2001)
10. Reyes Álamo, J.M., Wong, J., Babbitt, R., Chang, C.: MISS: Medicine Information Support System in the Smart Home Environment. In: Helal, S., Mitra, S., Wong, J., Chang, C.K., Mokhtari, M. (eds.) ICOST 2008. LNCS, vol. 5120, pp. 185–199. Springer, Heidelberg (2008)
11. Reyes Álamo, J.M., Yang, H., Wong, J., Babbitt, R., Chang, C.: Support for Medication Safety and Compliance in Smart Home Environments. International Journal of Advanced Pervasive and Ubiquitous Computing 1, 42–60