2010

A framework for safe composition of heterogeneous SOA services in a pervasive computing environment with resource constraints

Jose Manuel Reyes Alamo
Iowa State University, jmreyes@cs.iastate.edu

A framework for safe composition of heterogeneous SOA services in a pervasive computing environment with resource constraints

by

José M. Reyes Álamo

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Johnny Wong, Co-Major Professor
Carl Chang, Co-Major Professor
Samik Basu
Ying Cai
Wensheng Zhang

Iowa State University

Ames, Iowa

2010

DEDICATION

To God and to my family.

Table of Content

LIST OF TABLES

LIST OF FIGURES

ACKNOWLDGEMENTS

To my major professors: Dr. Johnny Wong and Dr. Carl K. Chang

The rest of my committee members: Dr. Samik Basu, Dr. Wensheng Zhang and Dr. Ying Cai

Collaborators in computer science department: Dr. Hen-I Yang, Linda Dutton and Dr. Simanta Mitra

To my lab mates: Tanmoy Sarkar, Ryan Babbitt, Fuchao Zhou, Ruchita Sirkanungo and Guillermo Hernández

Staff in the Graduate College: Thelma Harding and Dr. George Jackson

My undergraduate professors who helped through the process of going to graduate school: Yolanda Vélez, Edward Caro, José Juan Díaz Caballero and Fernando Santos

To my family: Luz Myriam Álamo Salgado, Ángel F. Reyes Fuentes, Ángel F. Reyes Álamo, Diana M. Rodríguez Escalante, Felícita Salgado Ramos and Suheyris Reyes Álamo

ABSTRACT

The Service Oriented Computing (SOC) paradigm, defines services as software artifacts whose implementations are separated from their specifications. Application developers rely on services to simplify the design, reduce the development time and cost. Within the SOC paradigm, different Service Oriented Architectures (SOAs) have been developed. These different SOAs provide platform independence, programming-language independence, defined standards, and network support. Even when different SOAs follow the same SOC principles, in practice it is difficult to compose services from heterogeneous architectures. Automatic the process of composition of services from heterogeneous SOAs is not a trivial task.

Current composition tools usually focus on a single SOA, while others do not provide mechanisms for ensuring safety of composite services and their interactions. Given that some services might perform critical operations or manage sensitive data, defining safety for services and checking for compliance is crucial. This work proposes and workflow specification language for composite services that is SOA-independent. It also presents a framework for automatic composition of services of heterogeneous SOAs, supporting web services (WS) and OSGi services as an example. It integrates formal software analysis methods to ensure the safety of composite services and their interactions. Experiments are conducted to study the performance of the composite service generated automatically by the framework with composite services using current composition methods. We use as an example a smart home composite service for the management of medicines, deployed in a regular and in a resource-constrained network environment.

CHAPTER 1. INTRODUCTION

A Smart Home is a pervasive computing environment equipped with technology such as sensors, actuators, services, and applications to assist the residents to perform their activities of daily living [1]. Smart Home research has mainly focused on helping the elderly and persons with special needs to stay home longer and live more independently. In order to promote aging in-place these homes must be customizable and extendable. Every person and every home have a different set of requirements and preferences. This makes standalone applications infeasible to provide a customizable, comprehensive solution for a pervasive environment such as the Smart Home. A flexible, dynamic solution where features can be added and removed without disrupting the overall functionality of the Smart Home system is preferred.

A service-oriented approach provides the tools to address the problems found in pervasive computing environments such as the Smart Home. Over the years, there have been a number of studies on different devices and development approaches to support these dynamic environments. A service-oriented approach has been adopted for providing solutions to the challenges of developing Smart Home services and applications. The Service-Oriented Computing (SOC) paradigm delineates the guidelines for the development of services and service-oriented applications. Within SOC, several Service-Oriented Architectures (SOAs) have been proposed, developed, tested, and deployed. These SOAs have provided solutions to a diverse set of important issues. Nevertheless, each SOA operates independently and there is still a need for solutions that integrate them.

This dissertation proposes a solution for the design and development of services and applications in pervasive computing environments using Smart Homes as an example. We strongly believe and support a SOC based solution. Over the years, several SOAs such as Web Services, OSGi, Jini and UPnP have emerged. While these SOAs follow the same SOC paradigm, usually the services developed in different SOA are not compatible with each other. Several researchers have tried to compose services of heterogeneous SOAs, as task demonstrated not to be a trivial one. As SOC gains more adepts, there is a greater need for

supporting composite services that combines services of heterogeneous SOAs. Based on the SOC standard, services are seen as individual units based on the functionality they provide and not as pieces of software developed using a particular SOA. Unfortunately, in today's real world applications, SOC development is tightly coupled to the particular SOA of the services. Being able to create composite services independently of their particular SOA, will have an impact in the number of services that can be selected. This could result in an increase in flexibility, availability, and functionality of individual and composite services. This approach can also have the effect of increasing scalability, reduce the cost, and require less development time and learning curve.

In this work a composition framework that allows creating composite services independently of the SOA of the individual services is proposed, developed, and tested. The requirements for creating composite services independently of their particular architecture are elicited. A discussion on how to incorporate these requirements into the SOC standard is presented. A SOA-independent language to specify the workflow of composite services, inspired in technology currently under use is defined. The framework developed, composes the services using this SOA-independent language, and creates an implementation of the composite service that supports services from heterogeneous SOAs. This composition process is completely automatic and platform, language, and SOA independent. Services used during the composition process are included without requiring any changes to their original implementation and without translating services from one SOA to another. During the composition process, the framework automatically checks a set of safety properties of the composite service and the services interactions. The framework is designed with a selection algorithm that capitalizes on performance, as our work focuses on improving this non-functional requirement for composite services.

A mechanism is presented to ensure that these compositions of heterogeneous services satisfy the defined safety criteria. To perform these checks formal software analysis techniques are used, specifically model checking. Different model checking techniques are incorporated as a fundamental part of our composition framework. Based on the services properties, they are categorized as baseline or extended. These different categories of

services supported and their interactions are checked. A semi-automatic technique is shown on how to model and check the baseline services that are assumed to run continuously in the system and to have a custom set of safety criteria. An automatic technique is used for extended services that employs the workflow specification of the composite service and automatically checks the composite services and their interactions. Extended services are assumed to be more dynamic and might come and go or change.

Our composition framework is designed focusing on the non-functional requirements of safety and performance. Different experiments take place to test whether our framework indeed satisfies these requirements. We are especially interested in performance as pervasive environment relying on services such as a Smart Home might be located in rural areas where resources like networking might be scarce. Performance of the composition process, the execution of the resulting composite service and of the communications becomes critical. To test our composition framework performance, the execution of composite services is measured under different network environments. The network environments used are a regular LAN network and a wireless mesh network (WMN). A WMN provides a constrained networking environment. In the experiments, the overhead that the composition framework adds to create the intended composite service and perform the model check is measured. Composite services using current composition techniques are created. These composites are usually generated manually and they use only services of the same SOA. The performance of the composite service automatically produced by the composition framework is compared with the performance of the composite service created using current methods. These different composites are executed under a regular LAN and WMN networking environment and a summary of the results and conclusions is presented. In the next section, a summary of the main topics relevant to this dissertation are presented.

1.1 Service-Oriented Computing

The properties of services make them an appropriate choice for development of pervasive computing environments like the Smart Home. A service is defined as an autonomous, loosely coupled, platform independent entity that can be described, published, discovered, and invoked. A service can perform simple or complex computations [2]. A service can be

atomic or can combine several atomic services into a composite. Every service belongs to a particular Service-Oriented Architecture (SOA). There are several SOAs like Web Services and OSGi. Each SOA has their particular set of features and strengths but all of them follow the same SOC paradigm. We believe that having a pool of services from heterogeneous SOAs available will help to better satisfy our performance and safety requirements, as the most appropriate services can be chosen without limiting to those of a single SOA.

In this work, two SOAs widely used in Smart Home environments are studied: Web Services and OSGi services. Web Services (WS) are software systems identified by a URL that use XML files to define their public interfaces and binding information. This allows other software systems to search and discover services definitions. These other systems may interact with the WS as described by its definition, using XML-based messages conveyed by Internet protocols [3]. WS are a very interesting SOA as a significant amount of research has been devoted to this area. This resulted in the development of a plethora of WS that can certainly be used. The other SOA studied is OSGi, which enables the deployment of services over wide area networks to local networks and devices. It specifies a layer with a common architecture for the execution of services. This architecture maps onto the physical and logical components providing a service platform that service providers can use to deliver services to customers in their own environment [4]. Pervasive environments such as the Smart Home make use OSGi, as it is very convenient for controlling small and embedded devices such as cell phones, sensors, and actuators. In addition, OSGi services are local and the framework where services are deployed has strong security.

There have been several standardization efforts for individual SOA, but not for combining services of different SOA. The Open Service Oriented Architecture (OSOA) [5] is an informal group of industry leaders that share the common interest of defining a language-neutral programming model that meets the needs of enterprise developers who develop software that exploits Service Oriented Architecture characteristics and benefits. Their initiative gears toward defining specifications for Web Services technology. Control over this effort has been transferred to the standardization group OASIS-OPEN [6], which is a

consortium that drives the development, convergence, and adoption of open standards for the global information society. IEEE apparently has the intention to standardize SOA, but little information can be found in their main portal for that effort [7]. The Open Services Gateway initiative (OSGi) alliance is responsible for the standardization of the OSGi framework and the services that the framework accepts [8]. These bodies, especially OASIS-OPEN have standardized several protocols for SOAs such as the SOAP, WSDL, WS-BPEL, and several other WS languages such as WS-Context, WS-Security, and WS-Reliable Messaging. The OSGi alliance has standardized the OSGi framework and the bundles structure, where a bundle is the mechanism used to package services. As noticed, each SOA has taken an independent approach into standardizing its own set of protocols. In our work, we present a framework that allows combination of services of heterogeneous SOAs and propose the guidelines for standardizing these kinds of operations.

1.2 Composition of Services

Several standardization bodies have proposed standards for composition of services. The OASIS-OPEN has defined the Reference Model for Service Oriented Architecture [9] to encourage the continued growth of different and specialized SOA implementations while preserving a common layer of understanding about what SOA is. In their Reference Model, they define the techniques for composition of services. They define the main technique for compositions known as orchestration as “a technique used to compose hierarchical and self-contained service-oriented business processes that are executed and coordinated by a single agent acting in a ‘conductor’ role.” The implementations of orchestrations are typically described using a scripting language. An orchestration engine accepts the scripting language and executes the process orchestration description. The orchestration engine is a hardware or software component that acts as the central coordinator for executing the flows that comprise the orchestration. The standard assumes there is a hardware or software component that executes the process flow, but does not provide specific requirements for these engines. The OASIS-OPEN definition is very high level and we believe that a minimum set of requirements and capabilities for these scripting languages used for orchestrations should be provided.

Several approaches deal with the problem of service composition for a specific SOA. In the literature, we find that most works have focused on strategies for composing WS [3]. Other approaches have tackled the problem of combining different SOAs. Some middleware solutions translate legacy code, a particular programming language, or a service into another SOA, usually WS [10]. After the translation process, WS composition techniques are used. The problem with this type of approach is the overhead incurred in the translation process and the extra layer of communication that WS imposes that can affect the performance. In addition, we believe that analyst, developers, and other stakeholders made important design decisions when they chose a particular SOA over the others. Therefore, the original services implementation and architecture should be preserved, which is one of the features of our framework. Other composition frameworks mentioned in literature rely on abstract representations of services and simulations [11]. In our work, we provide an example of an extensible framework that uses actual services and performs composition of heterogeneous SOAs. A guideline on how to extend it to support other SOAs is provided as well.

The Service Component Architecture (SCA) [12] is another standardized approach for the composition of service of heterogeneous platforms. The SCA specifications define how to create components and how to combine those components to form applications. The components of an SCA application can be built using technologies such as Java, C++, WS, Spring, and other services and programming languages that comply with SCA-defined programming models. SCA allows combining services developed using different technologies, but they have to follow the SCA guidelines. Services have to include a series of annotations in their source code for the SCA framework to accept them. This makes it difficult to use services their original implementation. In SCA, services bindings need to specify the particular technology or programming language of the service for the framework to be able to interact with it. The SCA standard assumes that services are deployed under the same domain and that entities will use the same vendor SCA framework implementation within their domain. Therefore, these set of assumption and requirements make SCA a tightly coupled technology.

A comparison of these frameworks and standards shows that none of them offers an independent, loosely coupled composition strategy that support services of heterogeneous SOAs. Our framework provides solution to these problems and considers the non-functional requirements of performance and safety. A motivating scenario applicable to our work in the future is in the deployment of interconnected smart homes in rural areas forming a smart village. Smart villages in rural areas might have limited network resources where efficient ways to compose services using minimum resources while ensuring safety becomes necessary.

1.3 Model Checking Composite Services

In this dissertation, formal software analysis, specifically model checking techniques are used to ensure the safety of the composite services and their interactions. Formal software analysis is a mathematically well-founded automated technique to reason about the behavior of a software system, with respect to a specification of soundness and unsoundness behavior [13]. Model checking is one of such techniques whose use in the last few years has increased as software becomes more complex and intractable. Model checking is a mechanism that helps to ensure that given a model of a system and a set of properties, whether the model satisfies the properties.

In this work, the proposed framework for composition of heterogeneous SOAs services pays special attention to the safety of composite services and their interactions. In the previous sections, a description of the strategy to satisfy performance requirements used by our composition framework is outlined. For satisfying the safety requirements, the composite services and their interactions are checked for compliance with the safety criteria. The composition framework automatically models the composite service specification and their interactions. It uses model checking techniques to verify whether the composite service satisfies the established safety criteria. Model checking is a fundamental part of our framework as this checking is performed before creating the actual implementation of the composite service. If the model satisfies the safety criteria, the framework creates the composite service implementation. If the model does not satisfy the safety criteria, it is rejected, the user is informed of the errors found, and the implementation of the composite

does not proceed. Safety checking is an important issue for service compositions in general and for those services created automatically.

1.4 Research Goals and Contributions:

We believe that the SOA standard should include a minimum set of requirements of supported capabilities for the scripting languages used for orchestrations. A set of these requirements and a justification for them is provided. The Simple Service Composition Language (SSCL) is defined, as a language compliant with these minimum set of requirements established for orchestration languages. SSCL is an XML-based language that is used to specify the workflow of composite services in a simple way that is SOA-independent. SSCL is simple to learn but with the expressive power to perform the functionalities required for services compositions. It is inspired in two XML-languages: the Business Processing Execution Language for WS (WS-BPEL) and the Service Component Definition Language (SCDL) for SCA. WS-BPEL is an XML-based language for specifying workflows and orchestrations designed for WS. SCDL is an XML-based language for specifying components and composites in SCA. SSCL combines the simplicity of SCDL with the expressiveness of WS-BPEL and supports functionalities such as invocation of services, basic data types, and decision and looping structures.

Our study of the standardization efforts for SOA shows that WS, SCA, OSGi and other particular SOAs have taken an independent approach and there are no specific guidelines for composition of heterogeneous SOAs in a structured way. Some of the SOA main characteristics are the separation of the implementation from the specification and the ability to reuse and compose services. However, the guidelines for compositions are very high-level and we believe there is a need for them to be more specific. We strongly propose SSCL and its design principles to be part of any SOA standard.

The composition framework presented accepts a composite service workflow description in SSCL as input and automatically produces an implementation of the composite service that relies on services of heterogeneous SOAs. The composition framework automatically searches the services needed, performs a safety check, creates the composite service relying

on services from heterogeneous SOAs in their original implementation, deploys the composite service, and starts it.

A crucial step for our composition framework for automatically compose services of heterogeneous SOAs is to have a registry that stores the information of candidate services. Several existing registries were studied such as the Universal Description Discovery and Integration (UDDI) for WS and the OSCAR bundle repository for OSGi services. A customized service registry is provided that allows storing the information of different services and searching for them later.

We show examples of composite services created by our composition framework that relies on services of heterogeneous SOAs. Our framework currently supports OSGi and WS, with the capability to be expanded to accept other SOAs. The main example used to test our framework is a smart home composite service for the management of medicines.

The composition framework uses an algorithm to select the service that better satisfies our requirements when several services match the search criteria. This algorithm is context-aware as is uses the context information to choose first those services already in use within the context. If a service is not found within the context, it searches the customized services registry and selects the service with the best performance.

The performance and safety of the automatically created composite services are analyzed to determine if they satisfy our requirements. Several experiments are conducted where the results of the automatically generated composite service are compared with the results obtained from using current composition techniques.

The composite services and their interactions are modeled and checked to ensure compliance with the safety criteria. For the set of baseline services, examples that model the services using PROMELA and their safety criteria using Linear Temporal Logic (LTL) are shown. These models are checked for safety compliance using the popular model checker SPIN as an example.

The safety of the extended services, like those automatically created using the composition framework is also checked. The interactions between these services, as specified in their SSCL file description, are checked for safety properties such as controllability, absence of deadlocks, and false nodes. These checks are performed automatically and integrated into the composition framework. Only those services that pass all the safety check are automatically implemented.

We claim that SSCL has the same expressive power of other orchestration languages such as WS-BPEL. Our claim is justified by showing that SSCL and WS-BPEL have compatible semantics even when their syntax differs.

We claim that the design of SSCL allows us to generate an open Workflow Net (oWFN) model of the composite services and that a compatible WS-BPEL process will generate the same oWFN. We justify this claim that will allow using existing tools to analyze and check the safety properties of oWFN.

The performance of our composition framework is studied in regular and in a resource-constrained environment. The performance under a constrained environment is studied to measure the feasibility of our composition framework. The regular environment is a LAN network while the constrained environment used is a Wireless Mesh Network (WMN). The overhead and the performance results of services deployed under a regular LAN network are compared with those deployed under the WMN.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows:

- Chapter 2 presents the literature review on the topics: smart homes, web services, OSGi services, web services compositions, formal software analysis, and wireless mesh networks.

- Chapter 3 contains the details of a set of essential services precursors of the automatic composition framework. These services use the smart home as an example. Our findings have been published in several conference papers and chapter 3 reports them.

- Chapter 4 presents a modified version of a published journal paper that describes in details the Medicine Information Support System (MISS). The MISS system the main composite service example used throughout the discussions in this dissertation.

- Chapter 5 includes a modified version of a submitted journal paper that describes the Simple Service Composition Language (SSCL) and provides the details of our automatic composition framework supporting services of heterogeneous SOAs.

- Chapter 6 contains a modified version of a submitted journal paper that presents the details of strategy used to ensure the safety of services by combining model checking techniques and how they were integrated to our automatic composition framework.

- Chapter 7 explains the performance analysis results by comparing the automatically composite service created by the framework with current composition techniques. Execution time under a regular LAN network is compared with the execution under a resource constrained WMN.

- Chapter 8 summarizes the conclusions and future work.

CHAPTER 2. LITERATURE REVIEW

In this dissertation, we present a composition framework that supports heterogeneous SOAs that automatically creates the composite service implementation that capitalizes on performance and uses model checking to ensure its safety. This work presents a comprehensive solution to the problem of composition of services of different architectures. However, several related works exist in literature. This chapter provides background and related work on these topics like smart homes, web services, web services composition, OSGi, formal software analysis, and wireless mesh networks.

2.1 Smart Homes

A Smart Home is a house that integrates different technologies for assisting the residents on their daily and repetitive activities [14]. This general definition encompasses a broad range of different smart homes and their target populations. The technology used in the smart home can assists the residents in performing their activities of daily living [15]. As a result, early work coined the concept of a health smart home [1]. The design of a health smart home especially targets people suffering different pathologies, the handicapped, and the dependent elderly, generally forced to hospitalization or placed in a nursing home. As most smart home research has focused on health smart homes, in this work we will make use of term Smart Home to refer to them.

A Smart Home is an example of a pervasive computing environment. Pervasive computing is also known as ubiquitous computing in some literature works [16]. A pervasive computing environment integrates computation into everyday object and activities [17]. The technology used in these environments includes sensors, actuators, and applications that interact with different appliances and data. Sensors are hardware devices that sense phenomena such as room temperature and humidity. Actuators are devices that can perform an action and change the state of the environment. Sensors and actuators can be programmed individually or using rules and policies to activate actuators based on sensors data. These devices can also be integrated with other technologies and be connected to the Internet,

greatly expanding the possibilities on how Smart Homes can be used to help people to stay home longer. Nowadays with these and other advances on sensors and actuators technology, pervasive computing is achievable.

There is a growing need for efficient techniques to be able to program these pervasive spaces [18]. One of the main limitations found in today's pervasive spaces is that the interconnection of their devices is very ad-hoc creating issues like scalability, the need for numerous and repetitive tests, and difficulties to integrate new technology. These issues have raised the need for software and hardware middleware solutions [19]. It is especially important to decouple the programming part from the physical-space and to allow different devices to integrate easily into the pervasive space. Several software standards going in this direction have emerged such as UPnP [20], Jini [21], and OSGi [22].

Other researchers have focused on hardware-based solutions. They have studied how to provide computing capabilities to sensors, actuators, and appliances via “smart-plugs" or some other innovative interface [23]. These interfaces make possible the interaction between an application and the physical device it controls. The hardware devices usually have a software service that controls the sensors, actuators, or appliances attached to it. The purpose is to make the developers’ life easier as they can focus on the application functionality and leave tasks like wiring and physically connecting the sensors to other engineers. These software and hardware based solutions try to reduce the learning curve in order to eventually have other domain experts such as psychologist, physicians and the owner of the pervasive space to develop their own custom applications and control different devices without requiring them to learn the different programming models used to control these devices.

There are several Smart Home projects across the United States and the world. At Iowa State University, the smart home research lab [24] has been proactively investigating different topics on smart homes technologies, service-oriented solutions, and security and privacy. Another project is the Gator Tech Smart Home [25] developed by the University of Florida. This project focuses on overcoming the challenges of integrating new technology into a pervasive environment. For this purpose, they developed a middleware sensor platform

called the Atlas platform [26]. This platform consists of hardware nodes, firmware and software middleware. Each node consists of three layers: the processing layer, the communications layer, and device connection layer. These layers are swappable and they provide services that allow controlling devices via software services. This sensor middleware platform is useful for context representation as services can return concepts such as “open”, “humid”, or “72F” instead of raw data from sensor readings.

Another Smart Home project is the aware home at Georgia Tech [27]. This project is an environment designed to learn information about itself and its inhabitants. The aware home has two identical independent living spaces where inhabitants live in one floor while prototyping and experiments takes place in another floor. This is a project intended to help elderly people as well. They built a prototype lab that resembles a house enhanced with technologies such as RFID tags and readers, sensors, and video cameras. They put special emphasis on context understanding via sensors and ways on how the computers at home can make use of this context information. They also study how to develop better context-aware applications closing the gap between sensed data and the applications that use it. In the aware home, there are wearable computers and intelligent environments that can learn the patterns of the inhabitant and perform activities on their behalf. They deployed a smart floor that can detect the location of a person by creating a ground reaction force (GRF) profile. The GRF profile computes a pattern of the footsteps of a person and predict with a 90%accuracy rate who is the person walking. One example of a context-aware application they developed is the Finding Lost Objects. They attach RFID tags on the objects and track the location of the person. Using this information, they can estimate the last position of the object before it got lost. The aware home project has a human-centered component that supports social connection among elderly and their family members.

The University of Washington has a project called the assisted cognition project [28]. Their interdisciplinary effort aims at helping persons with the Alzheimer disease. They study the relationship between the physical limitations and the environment. They work under the premise that a person with Alzheimer disease might be able to perform certain tasks in a familiar environment but the same tasks would become very difficult under an unfamiliar

environment. Their goals are to (1) use sensors to get an individual's location and environment information (2) learn and interpret their every-day behavior (3) offer help. Their interdisciplinary approach have the following advantages: (1) they have real data from real patients not relying on simulations (2) they collaborate with experts on elderly care (3) they test their system with a real population. They present two examples of assisted cognition systems. The first one is the activity compass, that is a tool to help direct a disoriented person to their destination using GPS and indoor sensors. They use this data to construct a high-level model of the person's activities. It does a plan recognition that determines if a deviation from the normal behavior should be interfered or not. Their second application is the adaptive prompter that is inspired in the fact that an Alzheimer patient might be unable to perform complex tasks but can perform simple tasks successfully. The prompter guides the person to perform a set of simple tasks that aggregated makes the complex task. They do it by capturing the sensor's data and detecting what the patient is doing then model and predict what the patient is trying to do, and offer visual or audible assistance when necessary.

2.2 Web Services

A Web Service (WS) is a platform-independent, loosely coupled, self-contained, programmable, web-enabled application that can be described, published, discovered, coordinated and configured using XML artifacts for the purpose of developing distributed interoperable applications [29]. The idea of WS is to provide the means so that different clients under different platforms can use the application provided. WS follow the SOA paradigm by separating the service specification from the service implementation. WS uses XML for specifying its service interface. The transport protocol used by WS is HTTP. By using these protocols, WS achieve platform-independence. Any programming language that supports WS can provide the service implementation. WS uses different XML protocols to describe it public interfaces and bindings, with the more important of these being SOAP, WSDL, and UDDI [3]. The OASIS consortium is in charge of driving and overseeing these WS protocols. More details on these protocols are discussed below.

The first fundamental protocol of WS is the Simple Object Access Protocol (SOAP) [29]. WS exchange structured information using the SOAP protocol messages. SOAP uses special

XML tags to format the content of these messages. A SOAP message is essentially an XML document consisting of three components: the envelope, the header, and the body. The envelope describes what is in the message and how to process it. The header contains processing and control information. The body contains the actual message.

The second fundamental WS language is the Web Service Description Language (WSDL) [29]. WSDL describes WS as a collection of communication end-points that can exchange messages. It details the interfaces exposed by the WS, the operations supported, the data exchanged, and how to bind and access the WS. An XML specification schema defines the WSDL. A WSDL file is to a WS analogously what an interface is to a class in an object-oriented programming language like Java. Several proxy code generators are available that takes as input WSDL files and produce source code for interacting with the service in the programming language used by the developer. These proxy code generators produce client code according to the service requirements and specifications as found in the WSDL file.

The third fundamental protocol in the WS standard is the Universal Description, Discovery, and Integration (UDDI). This protocol offers a centralized registry for services that allow users to search and find them based on certain criteria. The literature compares UDDI repositories with the way a phone directory works. UDDI provides two basic specifications: the definition of what information to provide about each WS and an API to query and update the registry. The UDDI Business Registry (UBR) was an initiative to have a centralized repository where every institution could access, register, and search for all kinds of WS. Corporations like IBM, Microsoft, and SAP sponsored this initiative but it is no longer in service. Instead, institutions now have to create their own private UDDI registries. There exists several WS search engines websites but they do not support a UBR-like registry [30],[31].

Other languages and protocols exist that provide additional WS functionalities. These additional specifications provide different features and satisfy other needs that the main protocols SOAP, WSDL, and UDDI do not. Some of these additional specifications are WS-Security, WS-Policy, WS-Notification, WS- Transfer, WS-Discovery, WS-BPEL, WS-

Coordination, and WS-Management [32]. These other protocols are also XML-based and they are contained in the body of a SOAP message. Different needs and requirements motivate the use of these additional protocols. One example of a widely used protocol is the Business Processing Execution Language for Web Services (WS-BPEL). This language has become the de facto standard for compositions of WS. We study WS-BPEL and compositions based on this protocol in more detail throughout this work.

Several tools are available to experiment with WS and manipulate these protocols and languages in a more user-friendly way. For the Java Developer, Eclipse has plug-ins such as the Web-Tools Platform (WTP) that allows the developers to work with WS using a Graphical User Interface [33]. As mentioned above the UBR is no longer available [34], but several open-source UDDI registry servers are available such as Apache jUUDI [35] and openuddi [36]. There are also APIs for manipulating UDDI such as UDDI4J [37].

For transmitting the XML-based protocols of SOAP, WSDL, UDDI, and others, the WS standard uses HTTP as its transport protocol. The choices of XML for defining the language and HTTP for transport, give these protocols advantages such as versatility, extensibility, and platform and language independence. The choice of XML as the basis for WS languages is appropriate, as XML is a widely accepted language, there are a number of tools available for parsing these files, it is lightweight, and it is firewall friendly. HTTP is the transport protocol used because it works well with the current Internet infrastructure.

2.3 OSGi Services

The Open Service Gateway Initiative (OSGi) is a consortium of more than 80 companies around the world whose mission is to enable the deployment of services over wide area networks to local networks and devices [4]. It specifies a layer with a common architecture for the executions of the services. This architecture maps onto the physical and logical components providing a service platform that service providers can use to deliver services to customers in their own environment. Some of the benefits of OSGi include platform independence, application independence, multiple services support, services collaboration support, security among different services from different providers, multiple network

technology, and simplicity. OSGi provides a new category of applications. It can connect to the Internet and execute services locally therefore certain services can use data from in-home attached devices as input. The architecture of OSGi consists of a service provider, gateway operators, Internet access, local network, and the devices. Some use cases for OSGi includes personal communications, energy management, security systems, alarm systems, health care, entertainment, information management, pay-per-use and synergistic services [38].

The OSGi main feature is its definition of a general-purpose, secure, managed Java virtual machine framework that supports the deployment of bundles. In OSGi context, a bundle is a set of resources all packed in a JAR file. Some resources are required while others are optional. The required resources are a special class called bundle activator, a manifest file, a Java service interface, and Java classes implementing the service. Additional resources include dependencies, other Java packages, and documentation. OSGi compliant devices can download, install, and remove bundles. Bundles can register services, import, or export Java packages. The framework is responsible for managing bundle related activities such as resolving dependencies, installing, starting, stopping, and updating them. The OSGi framework also provides a consistent programming model for bundle developers by decoupling the service specification from its implementation following the SOC paradigm and defining a SOA. Some of the basic OSGi services that the OSGi framework provides are the log service, the HTTP server, device access, configuration management, user management, and preferences [39]. Several commercial and open-source implementations of the OSGi framework are available such as Knopflerfish [40], Oscar [41], Eclipse Equinox [42], and Apache Felix [43].

OSGi is very suitable for programming smart environments such as the Smart Home because of its SOA support and its enhanced management of embedded devices. In literature, researchers prefer to use OSGi in this type of environment. For example, T. Gu et. al. in [44] provide a framework for developing context-aware application for pervasive computing using OSGi within Smart Home environments. They propose an ontology-based context model that leverages Semantic Web technologies and OWL (Web Ontology Language). They put it all together with a service oriented context-aware middleware (SOCAM) architecture

that includes services for context discovery, acquisition, and interpretation. Their key feature is their ability to reason about various contexts by using ontologies and first order calculus to describe formally the concepts in a particular domain. There are existing pervasive spaces that rely on OSGi technology such as the Gator Tech Smart Home [25] described previously.

2.4 Services Composition

Service composition is at the heart of the SOC paradigm. Most research in the service composition area has focused on WS, leading to the development of different approaches to accomplish this [3]. Some of the issues that arise with WS composition approaches are coordination, transaction, context, conversation handling, execution monitoring, and infrastructure. There are different scenarios that will motivate developers to perform service compositions. There are also a diverse set of composition methodologies that can be used such as static, dynamic, model-driven, declarative, automated, manual, and context based web service discovery. These issues and the different composition strategies have lead to the development of various service composition frameworks some of them manual, others semi-automatic or fully automatic. Examples of these frameworks are e-flow, MAIS, MOEM, SELF-SERV, OntoMat-Service, SHOP2, WebTransact, and StarWSCoP. Unfortunately, no single framework solves all the issues, satisfies all the needs for automatic composition of WS, or provides all possible composition mechanism. Most of these frameworks work individually and focuses on a particular issue or technique reason why a comprehensive solution still needs to be developed [3].

Different approaches have been proposed and studied in order to achieve automated WS compositions. WS composition is an active research topic, as atomic WS might not satisfy certain needs but when combined with other services in a certain logical way, the composite service does. At this point, WS composition is beyond the human capability for several factors that affect the tractability of services such as the large number of services available, the dynamicity of services, and the increasing number of organizations developing different WS. Automating this process is what has motivated the development of different composition strategies.

Two examples of composition strategies are the use of workflows [45] and AI planning [46]. With the workflow approach, the system uses a provided workflow to locate those services that satisfies it. Several platforms that use the workflow techniques are EFlow and Polymorphic Process Model (PPM). With the AI planning approach, the developer provides a set of constraints and preferences and the system generates the flow of the composite service and finds the candidate services. J. Rao and X. Su in [46] focus more on the AI planning technique but they mention some workflow details as well. They present a general framework for WS composition. They state that all the composition schemes should provide the following: presentation of a single service, translation of the languages (from WSDL to a formal, logic language), generation of the composition process model, evaluation of the composite service (to choose the best one out of several compositions), and execution of composite services. They point out that only WS language that directly connects with AI planning is DAML-S (also known as OWL-S). Different approaches for AI planning composition include situation calculus, Planning Domain Definition Languages (PDDL), rule-based planning, and theorem proving. Some of the tools available that support AI planning service composition are SWORD [47]and SHOP2 [11]. Most of these frameworks use abstractions of services in their compositions, while in this work a framework that works with real services is presented.

Automatic composition of OSGi service has been an active research topic of services as well. A. Wood et. al. presents in [48] a framework to achieve spontaneous compositions of OSGi services. Their framework provides functionality to handle availability of services, links to connect to the services and matching criteria for selecting the best available service. R. Redondo et. al. in [49] uses the WS-BPEL language to describe the workflow of a composition. They provide a framework that extends OSGi in order to support WS-BPEL files, locate the candidate services, and automatically create the OSGi bundle that implements the composite service. Nevertheless, their framework focuses on composing OSGi services only. J. Anke and C. Sell present in [50] a strategy for automatically composing OSGi services by first converting them to WS and afterwards use WS

composition techniques. Our work is different, as the services are used in their original implementation without the need for converting them to other architecture.

Other researchers have focused on the problem of composing heterogeneous SOAs. C. Lee et. al. in [51] uses WS-BPEL for composite service workflow description, but allow OSGi and WS combinations in the composite. In their work, the developer must clearly specify in the WS-BPEL file what type of service they intend to use (OSGi or WS) as well as the binding information such as the URL of the WSDL file. Our work is different as there is no need to specify the service type and binding information, as this information are automatically gathered from a service repository. We also define our own service composition language inspired on WS-BPEL, but ours is simpler and provides SOA-independence. The fact there is no need to provide the details about the particular services, allows our composition framework be extended to support dynamic bindings and heterogeneous SOAs. A similar approach is followed by technologies such technologies as the Service Composite Architecture (SCA) [51] found in frameworks such as Apache Tuscany [52], Fabric3 [53] and the Newton Framework [54]. The basic unit in SCA is a component that is a service developed using any supported SOA such as OSGi, Spring, and WS. SCA offers a framework in which developers can create composites using these components, developed under different platforms. Components use tags and annotation in their source code for intercommunication. To create composites, SCA use a special XML language Service Component Definition Language (SCDL), that takes the tags of the components and interconnects them. The problem with this technology is that the services code has to include these tags and annotations in order to work within the SCA framework. This limits the services that can be used to only those developed following the SCA specification. In our work, we strive at using the services in their original implementation without the need to change the source code or add tags or annotation, allowing reusability of services already available.

2.5 Formal Software Analysis

Formal software analysis is a mathematically well-founded automated technique to reason about the behavior of a software system with respect to a specification of soundness and unsoundness behavior. As software becomes more complex, using formal verification serves as an essential tool to ensure its correctness. There exist different techniques and trends in formal software analysis like model checking [13]. The technique of model checking consists of a model *M* of the system consisting of state transitions and a set of properties *P* to check. The checker will visit all possible paths of state transitions to see whether *P* holds on model *M*.

Other techniques to check formally the correctness of systems are abstract interpretation and the deductive method. Abstract interpretation instead of using real data it uses over approximation or under approximation. Abstraction of data is possible by looking at the sets and not the individual values. If two pieces of data belongs to the same set before an operation and after it, then this set can be used instead of the specific values. This greatly simplifies the model by reducing the range of values from where to choose to only values that represent entire sets [55]. The deductive methods [56] define the sets *P, C, Q* where *P* is a pre-condition, *C* is the program, and *Q* is the post condition. Given *P*, *C* and *Q* the checker verifies if running *C* with preconditions *P* results in *Q*. A disadvantage of this method is that it might require a lot of manual intervention to come up with the proper *P*, so that after executing *C*, results in *Q*.

The main problem of using formal methods techniques is the state explosion problems, where the number of possible states can increase rapidly. Several literature works deal with the state explosion problems and with infinite possible values of data by using different data taming techniques. One of this taming techniques is predicate abstraction [57] in which the values of the variables are mapped into a finite, smaller set of abstract values usually Boolean. The heap abstraction technique tries to reduce the number of nodes in a graph by moving into a single summarizing “super node” all those nodes that are not pointed by a variable in the system. The symbolic execution technique [58] executes the program using symbolic data and ranges instead of using actual data values. If the conditions are satisfied

the execution continues down that path, otherwise it stops. The partial order reduction technique [59] tries to eliminate interleaving among independent transaction in different threads that will not affect each other. Heuristic search technique tries to navigate faster a set of desired states. Another technique is the assume-guarantee, which analyzes that given an assumption, if certain property can be guaranteed. As noticed in literature we can find a lot of different approaches, tools, and future directions but rarely they evaluate their performance. Also because of particular implementation details what is a good configuration for a tool or a techniques it is not for the other.

There are several tool used for model checking that we are interested in using among them SPIN, Java Pathfinder and, FIONA. The SPIN model checker is a widely used model checker that perhaps is one of the most popular [60]. SPIN verifies systems modeled in PROcess MEta LAnguage (PROMELA) to see if they satisfy certain properties specified using linear temporal logic (LTL). SPIN also checks these PROMELA models for standard programming errors such as deadlocks and unhandled exceptions. There are several tools developed on top of SPIN, one of the Java Pathfinder (JPF) [48]. NASA developed JPF in their effort to produce error free code for their space, aviation, and robotics applications. JPF converts Java code into a PROMELA model, verifies a set of basic properties, and allows the developer to provide custom properties to check them as well. JPF suffer from the state explosion problem, posing limitations in the number of commands supported and the length of the programs it can translate. At this point, it only translates a handful set of Java commands and it runs efficiently with programs with up to a few thousands lines of code. Another model checking tool studied in this work is the BPEL2oWFN [61], which is a tool that converts a WS-BPEL files into open Work Flow Nets (oWFN). An oWFN is a special type of graph for modeling service processes such as those specified using WS-BPEL. The FIONA model checker [62], verifies properties of these oWFN such as controllability. Therefore, BPEL2oWFN can be used to produce the oWFN and Fiona to verify it.

Uses of model checking are explored to ensure compliance with system requirement and corresponding laws. One of the main contributions of this work is a framework for automatically create composite services of heterogeneous SOAs. Some of these services

handle sensitive data or control delicate systems. We need to ensure that these services function properly, that their interactions are safe, and that they comply with the appropriate laws such as those for privacy of personal and medical data. One of the laws in the United States that a Smart Home system must comply with when handling sensitive data is the Health Insurance Portability and Accountability Act (HIPAA) [13]. In literature, there are different efforts to define this and other laws using formal languages so that a computer can interpret it and check it [63]. This process is known as formalizing the law and is a necessary step as sometimes the language of the law can be ambiguous or subject to interpretation by a law, or medical expert. Such ambiguities make it difficult for computer systems to check for compliance. However, in our work an effort is made to ensure that our Smart Home does comply.

2.6 Wireless Mesh Networks

A wireless mesh network (WMN) is a dynamically self-organized and self-configured network with the nodes automatically establishing an ad hoc connection and maintaining the mesh connectivity [60]. Some advantages of WMN are low up-front cost, easy network maintenance, robustness, reliable service coverage, and compatibility with ad-hoc and other networks. Some WMN applications include broadband home networking, community networking, building automation, high speed MAN, and enterprise networking.

There are two types of nodes in a WMN: mesh routers and mesh clients. Wireless mesh routers have conventional routing capabilities plus support for mesh networking. The wireless mesh routers design allows them to have multi-hop communication, allowing the network coverage to be expanded. Wireless mesh routers have multiple wired and wireless network interfaces attached to them to communicate data among heterogeneous network protocols such as Ethernet, Wi-Fi, Bluetooth and Zigbee [64]. Wireless mesh router are assumed to have limited or no mobility at all. The nodes with limited mobility form the mesh backbone to which clients can connect.

A wireless mesh client is a network node with a simpler hardware and software platform. PDAs, cell phones, and laptop are examples of mesh nodes [65]. Sophisticated mesh clients

may also have routing capabilities. For example, a typical laptop equipped with Wi-Fi sends and receives data but does not route packets. A laptop enhanced with mesh routing capabilities will also route other client's packets, possibly expanding the network and strengthening the coverage. These mesh client with multiple network interfaces diversify the capabilities of ad-hoc networks.

There are three types of WMN: the infrastructure also known as backbone, the client, and the hybrid WMN. The infrastructure or backbone WMN is a network with a set of wireless mesh routers that have limited mobility and to whom clients connect. A Client WMN is more like a conventional ad-hoc network, where clients connect in a peer-to-peer fashion. Their main difference is that as part of the end-user requirements client nodes must perform routing and self-configuration functions. The third type of WMN is the Hybrid one that is perhaps the most popular one. The advantages of the hybrid design are that mesh routers and mesh clients can both perform routing functionalities, improve connectivity and coverage, and support ad-hoc networking with self-forming, self-healing, and self-organizing capabilities. The hybrid WMN forms the multi-hop wireless connectivity by having a backbone of wireless mesh routers that provide stability to the network. The end-nodes that may have mobility have the capability of increasing the coverage area and expand the network. Some of the challenges with WMN, especially the Hybrid, are scalability and performance. These problems are especially noticeable as the number of nodes in the network or the number of hops used to transmit a message increases. An interesting property of WMN is that they are not a stand-alone nor a new type of network, rather they are compatible with current wireless networks but combines them into a single infrastructure. WMN are used in our experiments to test the performance of our services and of the composition framework, as WMN provides an infrastructure where network resources are constrained.

CHAPTER 3. FUNDAMENTAL SERVICES IN PERVASIVE ENVIRONMENTS

3.1 Introduction

Throughout our research, we have studied several strategies and solution in Service-Oriented computing, software middleware solutions and services for pervasive environments like the smart home. Our finding such as requirements elicitation, design, development, testing, and results have been published in a series of conference papers over the last few years. This chapter provides modified versions of these papers presented in the order that our work progressed over time.

The paper entitled “MISS: Medicine Information Support System in the Smart Home Environment” [66] outlines a medicines management system that integrates the doctor, the pharmacy and the smart home. MISS ensures safety of medications intake by checking conflicts among new prescriptions with previous medications, medical conditions, and food items. The design, the model of the conflicts and a prototyped implementation are provided. The paper entitled “Service-Oriented Middleware for Smart Home Application” [67] documents our first works with having a middleware solution for simplifying the development and deployment of new services. It focuses on providing software-based service-oriented solutions especially for managing sensors. A layered architecture is presented with different demos that take advantage of the service-oriented middleware. The paper entitled “Composition of Services for Notification in Smart Homes” [68] documents a comprehensive composite service for giving notifications using various communication means. Repetitive task for communications are bundled into services, which are then composed to form the notification service. Several examples that show how the composite service is used along with the architecture for composite services in general are presented. The paper entitled “Using Web Services for Medication Management in a Smart Home Environment” [69] extends our previous works mostly focused on a single SOA by introducing Web Services. This paper introduces the idea of combining services from heterogeneous SOAs, integrates current systems, and provides interoperability by using WS.

Sensitive data among subsystems is securely transferred by using secure WS for communication purposes as shown in the prototyped implementation.

These services, composites, and middleware solutions were the precursors of the heterogeneous SOAs composition framework presented in this thesis. Checking the correct services interaction and managing sensitive medical data guided us into the use of combined model checking techniques for checking safety, also presented in this thesis. Next, the modified version of MISS [66] is presented.

3.2 MISS: Medicine Information Support System in the Smart Home Environment

A paper published in Proceeding of the 6th International Conference on Smart Homes and Health Telematics, 2008

José M. Reyes Álamo, Johnny Wong, Ryan Babbitt, Carl Chang

3.2.1 Abstract.

The Smart Home uses different technology to facilitate the lives of the resident and is especially useful for assisting the elderly and persons with special needs. One area where this population would benefit is managing their prescribed medications. This section presents the Medicine Information Support System (MISS) which integrates the patient's information to assist with the prescriptions management. The system checks for conflicting medicines, health conditions and food items. The data generated is used to feed other subsystems in the Smart Home such as the reminder and medicine inventory. A formal model is introduced for conflicts checking. The three main entities: doctor, pharmacy and Smart Home use this model to detect their particular set of conflicts which ensures that conflicts involving the entire context will eventually be detected. The design uses this model as its basis for conflict checking. The prototyped implementation of the entire system is based on Java.

3.2.2 Introduction

The Smart Home is a house that integrates different technologies for facilitating the execution of daily tasks. Sensors and actuators play a major role in assisting in the automation of these tasks. Smart Homes, with modern technology designed especially for the elderly and persons with special needs, have been a research subject in the last few years. One main motivation for this research is that the baby boomer generation is reaching the retirement age and the need for assistance increases as they grow old.

One of the areas in which the elderly and persons with special needs would need assistance from the Smart Home is in their medicine intake management. Keeping up-to-date with the prescriptions can be challenging due to complicated medicine names, several simultaneous medications, similar instructions for medication intake for different medicines, and being aware of expiration dates and detecting conflicts.

We propose MISS: Medicine Information Support System in the Smart Home to address this issue. A step-by-step analysis of the process for getting prescriptions today is presented. This analysis helped identifying important requirements for our system. As a result we come up with a formal model which is used for detecting errors and conflicts among sets. These sets are the medicines, food items and patient's health conditions and the system checks for conflicts among those. This system generates data which can be used by other subsystem in the Smart Home. The reminders system which tells the patient when to take the medicines is one of these subsystems that benefits from MISS. Also MISS data is useful for preparing and updating a personalized calendar and providing individual assistance for each user at the time of taking the medicine.

There have been previous efforts for helping individuals with the management of their prescriptions. Some related work includes the Magic Medicine Cabinet (MMC) which is presented in [1]. In that project the author mentioned that today's smart devices are designed to perform the device tasks plus connecting to the Internet. This converts the appliance to a device similar to a personal computer which also can go online. MMC is equipped with a facial recognition software, RFID smart labels, vital signs monitor and voice synthesis. The

MMC assists the residents of the house by giving personalized reminders, detecting when a resident take the wrong medicine, and measure some vital signs. The MMC is a great idea but their product is not designed particularly for the needs of the elderly population. They also do not give details of how it interacts with the patient's pharmacy, doctors and health care providers even though the MMC claims it can. We bridge this gap by describing a system in which the Smart Home interacts with the patient's doctor, pharmacy and health care provider. Such a system will be useful for checking conflicts and errors in the process of dispatching medicines, will facilitate giving reminders, and will increase compliance with medication intake [7].

The Smart Medicine Cabinet [2, 3] and the Smart Box [4, 5, 6] extends the Magic Medicine Cabinet by using passive RFID technology and Bluetooth, to synchronize the state of the MMC using a cellular phone. The cellular phone contains information to be used to give reminders and to know the state of the medicine cabinet and its content. They assumed that the medicine containers have RFID tags and the Smart Medicine Cabinet (SMC) can be automatically updated. When the cell phone is within the SMC range, a synchronization phase takes place keeping user intervention to a minimum. Nevertheless this still requires the user to carry the cell phone to the synchronization area as well as to carry it to the pharmacy. Our system presents a simpler synchronization process from the patient's point of view that will not require any intervention from the patient. Also there will be no need of carrying any device such as a cell phone giving our system another advantage.

Technology available for automatic dispensing of prescribed pills can be found in [10, 11]. All these products have some common features. In these machines either the resident of the Smart Home or a caregiver has to load the automatic dispenser with the medications, enter the times that the medicine should be taken, remove the medicines after the system reminds the patient for taking them and repeat these actions for other prescriptions. The disadvantage is that these machines require a lot of manual action.

All these products has some outstanding features that facilitate the task of taking medicines [10, 11], ensuring that the right medicine is taken at the proper time [4, 6] and

giving reminders to the patient [8, 9]. Also these products facilitate to some extent the detection of errors during the process such as when the medicines are not taken. But these products by themselves do not address the need of managing medicine information as they still require a lot of manual input from the patient. In [13] a system which uses this technology for helping patients with dementia is presented. Our system is similar in which it uses available technology to help patients with their prescription intake. Our system is different as it involves the doctor, pharmacy and health care providers at early stages. Also we provide a formal model of the system to ensure it is safe, secure and correctly detects the conflicts. Also MISS presents a system in which the patient does not need to enter any data manually and can be integrated with existing reminder systems such as outlined in [13]. The following sections will describe more details of our system MISS.

3.2.3 Current Technology and System Requirements

Describing the process of a person who goes to the doctor and receives a prescription will help to identify important system requirements. This process can be broken down into the following steps:

1. The person visits the doctor.

2. The doctor prescribed some medicines.

3. The patient goes to the pharmacy and gets the prescribed medicines.

4. The patient goes home and intakes the medicines.

Medicines intake involve the following steps:

4.1 Wait for the next dosage time

4.2 Locate the medicine container

4.3 Open the container

4.4 Extract the appropriate amount of medicine

4.5 Intake the medicine

4.6 Close the prescription container

4.7 Return the container to the medicine cabinet

Several of these steps have been automated with existing products. For step 4.1 and 4.2 a Smart Box or Cabinet can help with the reminders and location of the medicines [1, 2, 4, 6]. Using automatic pill dispensers [10, 11] can help in opening the medicines and extracting the right amount as indicated in steps 4.3 and 4.4. These are some examples on how current technology can be used for the purpose of medicine intake but there is still room for improvement. One of the steps which can be further improved is 4.1 where the patient "waits" for the next dosage time. The elderly and persons with special needs might forget when the next dosage is [13]. A system which takes care of reminding them the time of the next dosage will help to increase compliance with medicine intake [7]. Another problem facing this population is to locate the medicine as required in step 4.2. It is possible that they do not remember where the medicine containers were placed last time. Therefore the patient will benefit from a system that will help locating the medicine containers. The automation of a reminder system will require input of the prescription's specific information to be able to give the proper reminder at appropriate time. To enable the location of medicines, a unique identifier for the prescription containers is needed. This will facilitate tracking the object within the Smart Home and distinguishing it from similar products. An efficient mechanism to detect and locate the medicine container is required.

Entering the medicine information manually by the resident or a health care provider for these systems is not feasible. Human errors and typos can occur. Therefore the details about the prescription such as the name, dosage, conflicting medicines, conflicting food, patient conditions and other warnings should be entered by an expert and automatically transferred into the Smart Home system. This way intervention from the resident is minimal so this feature is very important. All this automation will require a reliable system which will accurately compute the existence of conflicts among the prescribed medicines, the food items

available at the Smart Home, and the patient health conditions. This data must be accurate as it is expected to be used as input for other subsystems in the Smart Home. The design and the prototyped implementation of the model show how these tasks can be accomplished by our system requiring minimum intervention from the patient. The next section shows the MISS system and design.

3.2.4 MISS System and Design

This section has a high level description of the MISS system, it subsystems and their interactions with each other for managing the medicines and detecting conflicts. At a very high level the system should do the following: The patient visits the doctor and gets a prescription. The prescription details are inputted into the system at the doctor's office. The system will check for conflicting medicines and heath conditions based on the patient's record that the doctor has. The patient will indicate from which pharmacy the prescription will be picked up. The doctor then will make the prescription's data available to that particular pharmacy. The pharmacy prepares the prescription based on the doctor's prescription. The pharmacy will double check for conflicting medicines and health conditions based on the patient's record of the pharmacy. The patient picks up the medicines. The patient goes to the Smart Home and in a very convenient way, by scanning the RFID-enabled prescription containers into a RFID-reader, will indicate to the Smart Home system the presence of the new medicine. The Smart Home will update its medicine inventory database accordingly and makes a final check for conflicts among medicines, health conditions and food. MISS consists of three main subsystems: The Doctor Subsystem, the Pharmacy Subsystem and the Smart Home Subsystem. These subsystems are operated by four main actors which are: the doctor, the pharmacy, the patient and the Smart Home. A trusted third party medicine's database that defines the conflicts is also used. A diagram of the system is shown in Figure 3.1. The next paragraphs describe each subsystem design more detailed.

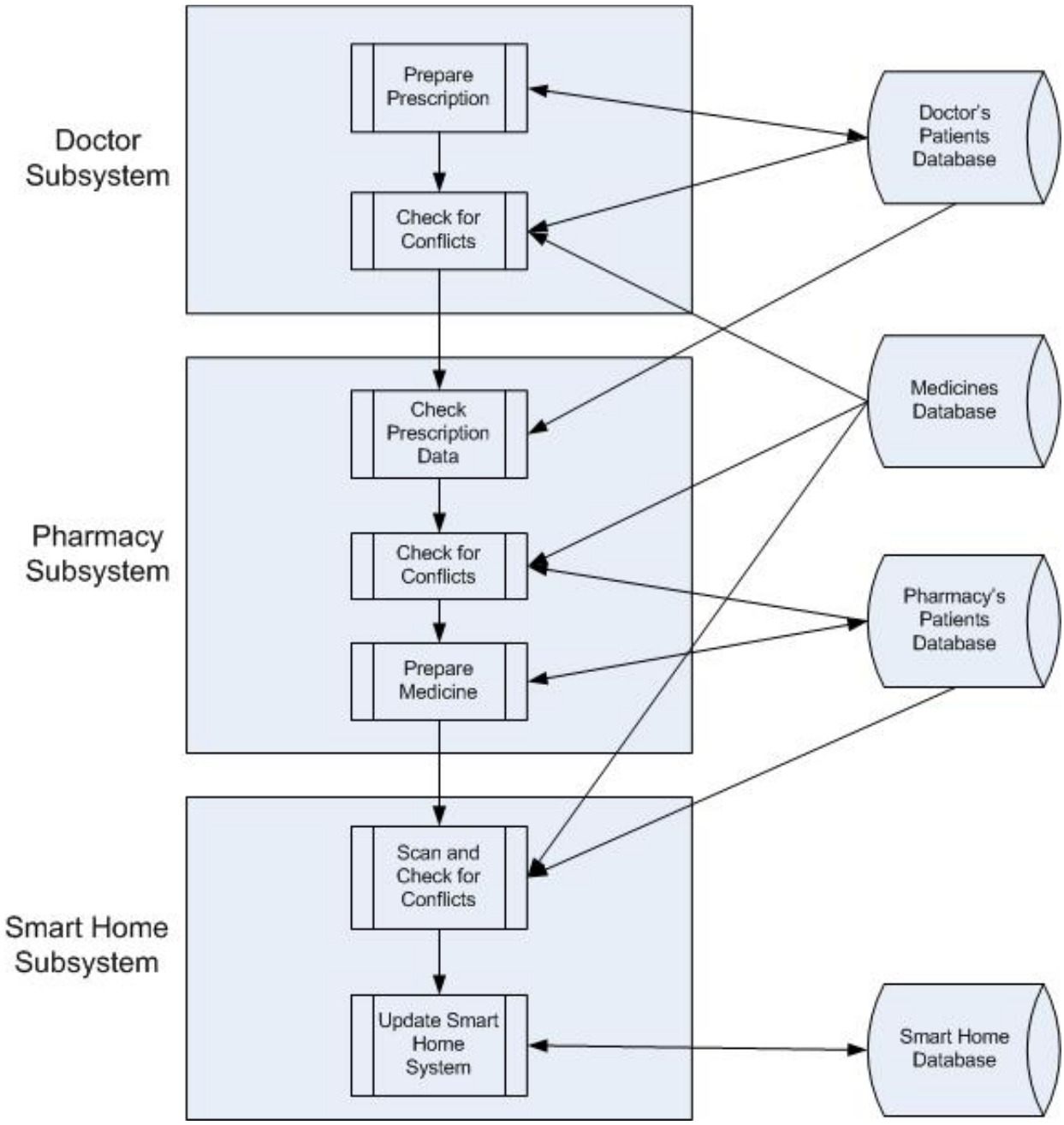

Figure 1 - Medical information system diagram

The doctor subsystem is where the process starts. During the visit to the doctor, he checks the patient and prescribes some medicine. During the consultation the prescription details such as the name of the medicine, dosage, etc. are inputted into the system. Our system checks for conflicts and also facilitates the communication between the doctor's office and the patient's preferred pharmacy. For doing this the options are to have the doctor's office directly communicate with the patient's preferred pharmacy and send the prescription data or making the prescription data available for a pharmacy that the patient will choose later.

Consider the first case when the patient chooses a preferred pharmacy to pick up the medicines. The person interacting with the system at the doctor's office will use a unique ID such as an assigned patient number, to access the patient's information. This information is stored at a local patient's database, available only to the doctor. The information extracted from the database contains data such as previous prescriptions and health conditions. To ensure that the prescription would not have any adverse side effect on the patient the information extracted from the database will be checked against the data of the new prescription. After carefully checking for conflicts with medicines and health conditions, if no conflict is found the prescription data will be sent to the patient's preferred pharmacy through a secure channel. The doctor's office will also issue prescription document customized for that particular patient and prescription. This prescription will be in the form of printed-RFID tag which will be used later by the pharmacy module.

In the second case in which the patient will decide later from which pharmacy the prescription will be picked up, the process is similar. At the doctor's office the prescription information will be entered into the system, the patient's data will be accessed and check for conflicting medicines and health conditions as in the previous case. But here the doctor's office will not be sending the data to any pharmacy. Instead a printed-RFID prescription will be issued. This will allow the pharmacy's system to download the prescription data from the doctor's office later on. The main difference among the two approaches is when the data arrives to the pharmacy. In the first case it arrives immediately in the second case when the patient gets into the pharmacy. Now let us consider how the pharmacy subsystem uses this data.

We now describe the pharmacy subsystem. Based on the doctor's subsystem operation we assume the following two starting scenarios for the pharmacy subsystem: There is a chosen pharmacy which receives the prescription's data from the doctor's office or the patient will choose a pharmacy later and bring the printed-RFID prescription.

Consider the first scenario in which the patient chose a preferred pharmacy at the doctor's office. The pharmacy will receive the prescription's data with the necessary details when the patient is still at the doctor's office. The pharmacist can start preparing the prescription immediately using this information. The pharmacy will issue the prescription in special containers. These containers will look like regular ones with the difference that they will be equipped with RFID tags. These RFID tags will allow the system to uniquely identify the particular medicine with all its related data. This tag-ID and the related information will be stored in a database that will be used later by the Smart Home module.

Before assigning the RFID tags for the container, the pharmacy system should have available a history of the patient and previous prescriptions dispatched from that pharmacy. The system will check that the particular RFID tag has not been assigned maintaining uniqueness. After assigning unique IDs to each prescription, the system is ready to prepare the data that will be used for updating the Smart Home subsystem. This data consists of two important pieces: the patient independent and the patient dependent information. The patient independent information contains the description of the medicine, possible side effects, different conflicts and recommendations. The patient dependent information is the historical data that the pharmacy has about the patient. This information will be checked in a similar way as it was checked in the doctor's module. The system will be looking for possible conflicts with other medications, and health conditions to make sure it is safe to take that particular medicine. This double check is necessary as the doctor might prescribe a medicine which creates a conflict with a medicine previously picked up at that pharmacy.

At this point the pharmacy is ready to receive the patient. When the patient arrives he shows the printed-RFID prescription the same way they are used to do it now. The difference is that the prescription ready or in process as the patient chose the preferred pharmacy when

still at the doctor's office. The printed-RFID prescription is scanned in a RFID reader. At this point the system will compare the data in the RFID tag with the data received from the doctor's subsystem. If no incongruence is found, the medicine is dispatched. Otherwise, the pharmacist is alerted and contacting the doctor's office is recommended. One advantage of our system is that it reduces the waiting time at the pharmacy as pharmacists can start preparing the prescription when the patient is still at the doctor's office. This is an excellent feature especially for the elderly population that might need their medicines as soon as possible or want to avoid long waits or several trips to the pharmacy. Another benefit of our system is the double layer of security checking for conflicting medications and health conditions.

In case that the patient did not choose a preferred pharmacy the waiting time will increase but the process will be almost identical to the one described previously. Instead of having the prescription ready or in process, the patient starts the process of getting the prescriptions when arriving into the pharmacy. At the pharmacy counter the pharmacist will receive the printed-RFID prescription and at that moment the pharmacy system will download the prescription data from the doctor's office and perform all the safety checks for conflicting medicines and health conditions described in the previous paragraphs. At this point whether the patient pre-selected the pharmacy or not, the medicines should be ready and the patient can go home. The process for updating the system in the Smart Home will be very simple from the patient's point of view as the underlying system will take care of all the details.

At the Smart Home subsystem, the patient finally arrives to the Smart Home and updates the subsystem with the new prescription's data. Either the patient or a caregiver will be in charge of updating the Smart Home system by scanning each prescription container with an RFID reader. After scanning the prescriptions the medicines can be placed in Smart Medicine Cabinet [2, 4] or loaded into an automatic medicine dispenser [10, 11] or a combination of both technologies for storing the medicines.

At this point the Smart Home system will read the RFID-tags of the prescription container. These tags will contain information indicating from which pharmacy the patient

picked up the medicines. The Smart Home will have a secure communication link to the pharmacy. A query will be issued to the pharmacy to retrieve the prescription details and download this data to the Smart Home subsystem, similar to the process of downloading the data from the doctor to the pharmacy. A final safety check for conflicting medicines, health conditions and food items will take place. This check is necessary as the patient might be picking up the medicines from different pharmacies, prescribed by different doctor. The check for conflicting food will be performed at the Smart Home level as it is the subsystem with the database of available food items.

If a conflict is detected then a caregiver will be informed. If no conflict is detected then with all this information, the Smart Home System will update its medicine inventory and provide data to other subsystems such as the reminders and personalized calendar. All these tasks that MISS perform will definitely help the elderly and persons with special needs that have a hard time checking all these safety issues by themselves.

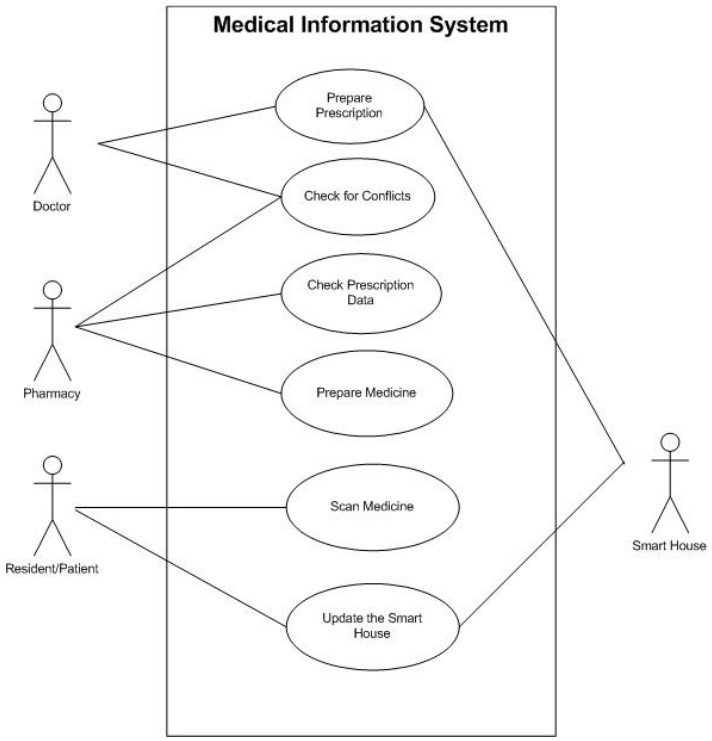

Figure 2 - Use cases and actors

The more important use cases and actors for the MISS system are shown in Figure 3.2. The system will have four main actors: the doctor, the pharmacy, the patient and the Smart Home. From the figure we can see that the doctor actor is in charge of starting the system by preparing the prescription and make it available to the pharmacy. The doctor actor will also be the first one to check for conflicting medications and health conditions. The pharmacy module then will use the information provided by the doctor's office to prepare the appropriate prescription. It will also check for conflicts and will make sure to validate that the data received from the doctor's office is correct. The patient actor will scan the RFID enabled prescription containers to feed the system with the information necessary to obtain the prescription details from the pharmacy's database. The Smart Home actor will then download the prescription's data from the pharmacy and will use it to update it medicine inventory and support the reminder, calendar, notification and location and tracking subsystems. To ensure the accuracy and correctness of the conflict detection a formal model is proposed in the next section.

3.2.5 Proposed Model

The medicine management system should be accurate, reliable and provide safety by detecting and informing conflicts. A conflict occurs when a medicine should not be taken together with another medicine, if the patient has certain health condition that the medicine could aggravate or the medicine interacts with certain food items. We are assuming that a trusted third party defines the conflicts and these definitions are publicly available for the model and the system to use them. To check that these kind of conflict do not occur we present the following model whose main components are the set of medicines M, the set of food items F and the set of medical conditions C. The set of food items F needs to be considered as some food should be avoided when taking certain medicine. The set of medical conditions C needs to be considered as some medicines cannot be taken if the patient has certain conditions.

Now we define several functions that will act over the set of medicines and return useful information. The first function is *conflicting_medicines:* $M \rightarrow P(M)$, where $P(M)$ is the powerset of the set M, which determines the set of conflicting medicines. The next function

is *conflicting_food: M→P(F)* where *P(F)* is the powerset of the set *F,* which will return the set of conflicting food items. The other function is *conflicting_conditions: M→P(C),* where *P(C)* is the powerset of the set *C,* which will return the set of conflicting health conditions. Given these sets and function we define the Medicine System Model as follows:

Definition

A Medicine System Model *S* consists of the following sets:

- M, the set of medicines
- C, the set of medical conditions
- F, the set of food items
- D, the doctors, hospitals or clinics the patient visits
- P, the set of pharmacies at which the patient gets prescriptions
- H, the patient's Smart Home

With the following functions:

- conflicting_medicines: M→P(M)
- conflicting_conditions: M→ P(C)
- conflicting_food: M→ P(F)

We want this information to be useful for a particular patient *p*. Each patient will be represented by a tuple.

p = (id, Mp, Cp, Fp, CMp, CFp, CCp). The *id* entry will uniquely identify the patient. Let $Mp \subseteq M$, represents the subset of medicines prescribed to patient p. Let $Cp \subseteq C$ represent the

health conditions that patient *p* has been diagnosed. Let $Fp \subseteq F$ represent the food items that patient *p* has available. Let *CMp* represent the subset of medicines that are currently in conflict with the medicines prescribed to patient *p* and therefore should not be prescribed to that patient. *CMp* can be computed as $\bigcup_{m \in Mp} conflicting\_medicines(m)$. *Let CCp* represents the set of medical conditions that the patient should not have in order to take the medicine safely. *CCp* can be computed as $\bigcup_{m \in Mp} conflicting\_conditions(m)$. Let *CFp* represents the set of food items that the patient should avoid while taking the medicines prescribed to him. *CFp* can be computed as $\bigcup_{m \in Mp} conflicting\_food(m)$. This will help to detect when a patient is diagnosed with a health condition and is taking a medicine which is in conflict, or when a medicine is prescribed which is in conflict with an existing health condition. This way the doctor or caregiver can make better decisions.

Now we describe how to construct these sets that complete the patient information. Given a new prescribed medicine *m*, to a patient *p*, the medicine has some data related to it like the unique drug id, active ingredients, milligrams, and so on. The medicine data can be obtained from the Food and Drug Administration (FDA) [14], or from the Physician's Desk Reference (PDR) [15]. These entities are trusted third parties who define the conflicts among medicines, food and health conditions. These definitions of conflicts will be used in our model and system. Using the data of medicine *m*, we compute $CM = conflicting\_medicines(m)$, which returns the set of conflicting medicines with medicine *m*. We check if there is a conflict by examining if $CM \cap Mp = \emptyset$ and $m \cap CMp = \emptyset$. If both conditions are true then we take $CMp = CMp \cup CM$, to update the set of conflicting medicines for patient *p*. We do a similar processing for the conflicting conditions. We compute the set $CC = conflicting\_conditions(m)$ which returns the set of conflicting conditions with that particular medicine. We then check for conflicts by taking $CC \cap Cp = \emptyset$. If condition is true, we update this information for the patient by computing $CCp = CCp \cup CC$. Similarly we compute the set $CF = conflicting\_food(m)$ which will return the set of conflicting food items for that particular medicine. We then check for conflicting food items by computing $CF \cap Fp = \emptyset$. If this set is empty then no conflict is found and an updated

version of the conflicting food items is computed $CFp = CFp \cup CF$. The next section will show how to check for conflicts at each component of the system

3.2.6 MISS Conflict Checking

MISS is composed of three subsystems: The Doctor Subsystem, the Pharmacy Subsystem and the Smart Home Subsystem. MISS will also access a global medicines database from a trusted third party. Each of these subsystems is responsible for checking for conflicts but it is expected that each one captures a more specific set of conflicts than the others. The algorithm for checking conflicts is very similar. Therefore we define the following two routines as follows:

Definition: Get Data (GD)

Input: Prescription *r = (p, m)*

//Get the data of *p* querying the local database

Query *Mp, Cp, Fp, CMp, CCp, CFp*

//Compute data of *m* from the global medicines database

Compute CM, CF, CC

Definition: Conflict Checking (CC)

1. Input: Prescription *r = (p, m)*
2. Call Get Data (r)
3. If ($CM \cap Mp = \emptyset$ and $m \cap CMp = \emptyset$)
4. If ($CC \cap Cp = \emptyset$)
5. If ($CF \cap Fp = \emptyset$)
6. //No Conflict found

7. $CMp = CMp \cup CM$

8. $CCp = CCp \cup CC$

9. $CFp = CFp \cup CF$

10. Else

11. Medicine *m* creates food conflict

12. Else

13. Medicine *m* creates a health condition conflict

14. Else

15. Medicine *m* creates a medicines conflict

In this model it is assumed that the process starts when the patient visits the doctor and is prescribed with some medicine. Therefore at the Doctor Subsystem *(DS)* the model is fed with a new prescription *r = (p, m)*. This prescription *r* contains the id of the patient *p* and the id of the prescribed medicine *m*. The DS will have a local database with information stored about the patient *p* such as previously prescribed medicines, and health conditions. The DS is assumed not to store any information about food, so this set will be empty. This means that in the DS checking for conflicting health conditions and conflicting medicines will be enforced. This check will be performed by invoking the previously defined function *CC* with input *r*. If no conflict is found then the prescription *r* is sent to the Pharmacy Subsystem for further checking.

At the Pharmacy Subsystem *(PS)* it is assumed that the prescription *r* has been checked at the *DS* and no conflict has been found. The *PS* therefore will receive the prescription *r* from the doctor. It will then use this information to further check for conflicts. It is expected that the *PS* will have a local database with the patient's record of previous prescriptions and over-the-counter medicines bought at that pharmacy. This data may be different to the one at the *DS.* It is possible that the patient is visiting different doctors and a different prescription

from different doctors might be the source of conflict. The patient also might buy over-the-counter medicines which could be the ones that create the conflict, so all of them must be checked. It is assumed that the pharmacy does not store information about food, so this set would be empty. Therefore the *PS* must check again for conflicting conditions and conflicting medicines but using the pharmacy's local data. This is performed by calling the previously defined function *CC* with input *r* using the pharmacy's dataset. If no conflicts are detected the data is clear to be sent to the Smart Home Subsystem.

The Smart Home Subsystem (*SS*) will receive the data from the *PS* in the form of a prescription *r*. The *SS* is expected to check for any remaining possible conflict among medicines such as those picked up at different pharmacies. We are expecting the *SS* to have a local database with an inventory of the medicines and the food items available. This will allow checking if there is any remaining medicines conflict. The patient might be visiting different pharmacies and different doctors. The medicines prescribed from different doctors might create a conflict at this should be detected at the pharmacy. But if the patient is also visiting different pharmacies, these conflicts can go undetected. These multiple paths of conflicts are the ones that the *SS* will be responsible of detecting. Also again the patient might buy over-the-counter medicine at a gas station or grocery store for example. These can create undetected conflicts as these stores are not part of our system. But when the patient arrives at the Smart Home, the *SS* have the capability of detecting these conflicts with over-the-counter medicines as well. The *SS* also detects any food items in conflict with the new prescription or with over-the-counter medicines. All these checks are performed by calling the function *CC* with input *r* using the *SS* local dataset which includes the medicines inventory of previous prescriptions and over-the-counter medicines, and the set of food items available.

3.2.7 Instantiation of the Model and System Design

Now that we have defined the main subsystems we want to ensure that everything works correctly and the system actually detect conflicts. For this instance imagine the following scenario. A patient visits the doctor and the doctor prescribes three medicines. The doctor records indicate a previously prescribed medicine and diagnosed health condition. One of the

three newly prescribed medications will create a conflict and this will be detected. Later the patient goes to the pharmacy, where he previously picked up a prescription from another doctor. The system should detect a conflict among the medicines prescribed by these two different doctors. Now the patient arrives home and only one of the three prescriptions so far has not find any conflict. But at the smart home the patient has some food item which should be avoided with that medicine and this is detected by the system. Based on the previous scenario we will present now an abstract instance of the model followed by an instance that uses real data of drug interactions pulled from the PDR Drug Interaction Tool [15].

Let's consider an abstract instance of the model. In *Table 1* we have the patient's data stored at the doctor's module in the row DS, the patient's data stored at the pharmacy's module in the row PS and the patient's data stored at the smart home module in the row SS. *Mp, CMp Cp, Fp* are as defined in Section 5. The medicines *MA*, *MB* and *MC* are prescribed by the doctor with the data as shown in *Table 2*. CM, CF, CC are as defined in section 5. In this instance we have that for prescribed medicine *MA*, the *DS* will detect a conflict with conditions *C1*, but medicines *MB* and *MC* will find no conflicts based on the doctor's data about the patient. When prescription data arrives to the pharmacy, the *PS* will detect a medicines conflict among prescribed medicine *MB* and previously prescribed medicine *M2*. No conflict is found with medicine *MC* up to this point. When the *SS* checks, it finds a food-drug conflict with prescribed medicine *MC* and food item *F3*. Therefore the patient or caregiver can be informed of this. This is an example on how the lack of information of the entire context can lead to a conflicting prescription, but with this system it will eventually be detected. The abstract model shows that this can be applied to any set of medicines. Also shows an example of the different paths that can create a conflict. In general a patient is seeing different doctors who prescribe different medicines. Prescriptions from one doctor might create a conflict with prescriptions from another doctor and the pharmacy module would detect that. But if the patient is also visiting different pharmacies these conflicts may go undetected. Also buying over-the-counter medicines at places different than the pharmacy can create the conflict. But at the Smart Home subsystem these conflicts would be detected as it works as a sink node. We want to ensure safety in this system which is one of the main

motivations to perform these conflict checks repeatedly from the very beginning of the process.

Consider now the case with real drug interaction data set pulled from the PDR Online Drug Interaction Tool. In Table 3 we have the data stored at the doctor's module in the row DS, the data stored at the pharmacy's module in the row PS and the data stored at the smart home module in the row SS. Table 4 shows the medicines prescribed by the doctor in this case Zoloft, Percocet and Allegra. The data about the conflicts is also presented in each column. At the doctor's module the patient's data indicates he was previously prescribed Ambien and has a condition of hallucinations. Therefore a conflict is detected with the prescribed medicine Zoloft which is not recommended if a patient has hallucination and also conflicts with Ambien. The rest of the prescription Percocet and Allegra find no conflict with Ambien. At the pharmacy the patient previously had a prescription of Xanax from a different doctor. Therefore a conflict with Percocet is detected as these two medicines should not be taken together. The medicine Allegra has found no conflict yet. Now at the Smart Home module we have the medicines inventory and the food inventory. A conflict is found between Allegra and Orange Juice as they should not be taken together. In the next subsections the prototyped implementation of this model are described.

	Mp	**CMp**	**Cp**	**Fp**
DS	M1	M11	***C1***	F1
PS	**M2**	***MB***	C2	F2
SS	M1, M2, M3	M11, MB, M33	C1, C2, C3	F1, F2, ***F3***

Table 3.1 Patient's data at each subsystem

M	**CM**	**CC**	**CF**
MA	M0	***C1***	F11
MB	***M2***	C22	F22
MC	M32	C33	***F3***

Table 3.2 Prescriptions at DS

	Mp	**CMp**	**CCp**	**Fp**
DS	***Ambien***	Zoloft	***Hallucinations***	*****
PS	Xanax	***Percocet***	*	*
SS	Ambien, Xanax	Ambien, Percocet	*	***Orange Juice***

Table 3.3 Patient's medicines at each subsystem

M	**CM**	**CC**	**CF**
Zoloft	***Ambien***	***Hallucinations***	*
Percocet	***Xanax***	*	*
Allegra	*	*	***Orange Juice***

Table 3.4 Example of a prescription at DS

3.2.8 Prototyped Implementation

The prototyped implementation of several use cases in our Smart Home Lab shows the feasibility of this system. The Doctor, Pharmacy and Smart Home Subsystems have been implemented as follows. One node has a customized application for inputting the prescription details and stores these in a database acting as the doctor's module. This node sends the data to another node which acts as a server with the patient dependent and patient independent information corresponding to the pharmacy. It assigns an RFID tag to the prescription

received from the doctor's module. Then we have another node acting as the client, querying the pharmacy's computer using an RFID tag as the key. We used a Phidget RFID reader [16] and assigned different RFIDs to several containers and tested the reading of tags. After reading the tag the client computer queries the pharmacy database and downloads the specific information that matches the RFID-tag as the primary key. The information includes details about the prescription. This data obeys a format that the Smart Home can store it in its database and use it to update other subsystems such as the reminder, notification and medicine inventory. When a conflict is detected the corresponding message is displayed. The Smart Home subsystem was developed as a bundle that runs is OSGi, a framework particularly suitable for Smart Home applications [17]. The flow of data was correctly transferred from the Doctor's node all the way to the Smart Home and the experimental conflicts were correctly detected.

3.2.9 Conclusions and Future Work

The management of medicines and prescriptions by the elderly and people with special needs might be a challenging task for this population. A system that reduces manual intervention and a model for checking and detecting conflicts is presented. This system starts with a visit to a doctor which enters the prescription information into the system and check for conflicting medicines and health conditions. This data is made available to the pharmacy, through RFID-enabled prescriptions. The pharmacy performs a double check for conflicting medicines and health conditions and prescriptions are dispatched in special RFID-enabled containers. The patient scans the special containers in the Smart Home which updates the system with the data it downloads from the pharmacy. The Smart Home checks for conflicting medicines, health conditions and food items and if no conflict is found updates the calendar, reminders, inventory and other subsystems. MISS correctly detects conflicts using a formal model and will facilitate the task of giving reminder and increasing medicine in-take compliance. Integrating this system with the whole medicine dataset provided by the official governmental and medical entities such as the FDA and PDR is on-going.

3.3 Service-Oriented Middleware for Smart Home Applications

Modified from a paper published in Proceedings of the IEEE Wireless Hive Networks Conference, 2008

José M. Reyes Álamo, Johnny Wong

3.3.1 Abstract

Smart Homes uses different devices including sensors and actuators to provide services that assist in performing the activities of daily living. Currently these smart homes are developed in a manual and adhoc way. Manually configured smart homes create scalability, extensibility and cost problem. The need for a uniform platform for intercommunication among devices and a middleware abstraction using service-oriented technology is presented. A layered architecture for smart home developers to use is presented. Different demos that take advantage of the service-oriented middleware layer are shown.

3.3.2 Introduction

The Smart Home is equipped with sensors, actuators and other technology to assist the resident performing the activities of daily living. Most of the research in smart homes focuses in assisting the elderly and persons with special needs [1]. The rich number of devices to use as well as the diversity of these technologies poses tremendous challenges at the time of development. Having all these devices working together is usually done in a manual, adhoc way [26]. Having a uniform way for developing and deploying smart homes applications becomes necessary in order to speed the process and reduce the cost. This uniformity will help ensuring that the smart home performs the tasks it is supposed to. It will also maintain the non-intrusiveness, making the resident feel comfortable within a technology pervasive space.

A smart home is implemented by combining and integrating technology already available. This allows providing a series of composed services that a single product cannot, adding more variety to the task that can be assisted by a smart home. Most services and

applications in a smart home need some type of sensor such as temperature, light, or weight. These sensors are passive and are used to monitor the environment and measure phenomena. Services may use actuators such as light controller, appliance controllers, and motors to automatic open or close doors. Actuators are active devices that can control other devices and change their state. Smart homes combine sensors, actuators and applications providing a variety of individual and aggregate services and making the smart home a hybrid sensor network. This interaction among devices does not necessarily seek to facilitate the communication among human-to-human or human-to-machine but the device-to-device.

This section shows how different sensors, actuators and application can be combined to make application for the smart home. It also shows the difficulties on integrating different technology into a single service. Based on these difficulties, important requirement are indentified to facilitate the development and deployment of such services and applications within a smart home. Several examples of current and proposed services and applications are explained. Based on the difficulties and the requirements found we proposed an architecture for smart home sensor applications especially sensor applications. This architecture has several layer one of them the software middleware layer. This software middleware is the key to improve development and interoperability among devices. Several applications on the smart homes that will benefit of using our proposed architecture and middleware are explained. Different demos that have been implemented using this service-oriented approach are also explained and one of these demos is explained more detailed.

3.3.3 Related work

A middleware solution for sensor networks have been studied in the past and some solutions have been proposed. For example [19], presents Atlas that is a middleware sensor platform. The idea of Atlas is to enable programmable pervasive spaces making the sensors plug-and-play. To accomplish this the sensors connect to an Atlas node, which is a piece of hardware, and the Atlas node has to be configured by indicating which sensors are connected to it. After this initial configuration the Atlas node is turned on and it will load a service to an OSGi framework and the sensors will be ready to use. This is a possible solution nevertheless using

Atlas adds another piece of hardware which requires a constant source of power. The sensors supported are analog sensors which require wiring. Also the configuration of the Atlas nodes is still very manual and does not detect when a sensor is changed, you have to re-configure the node. Using current standards such as IEEE 802.11 for sensor networks is proposed in [4]. This have the advantage that these protocols are well defined, thoroughly tested and allow for interoperation with current devices that uses the standard. Even thought devices that use IEEE 802.11 consume less energy than in the past, using this standard for sensor networks needs more investigation. In [70] it is proposed a set of configurable devices to use for health monitoring which can be controlled by a Pocket PC. The problems is that the battery lifetime is too short and that the Pocket PC needs to be carried all the time, even inside the smart home for this application to work.

As we can see several devices and application has been developed to alleviate the problem of using wireless sensors for different applications. These same techniques can be integrated together and used in the smart home but this process needs to be simplified and standardized. In the next few sections we define the smart home as a device-to-device environment.

3.3.4 Smart homes as a device-to-device environment

The smart home uses sensors, actuators and applications in order to assist the resident in performing activities of daily living. The emphasis is on the performance and communication among devices and application and not necessarily human-to-human or human-to-computer [71]. Therefore a smart home fits into the definition of a wireless hive network. We present two cases of applications that use different devices to provide a service and how smart home fits the wireless hive network definition.

Consider first the case of a smart home that assist the resident in the management of their medicines such as in [66]. This system integrates the doctor, pharmacy and the smart home and check prescriptions for conflicts with other medicines, food or medical conditions. In this system each prescription has a unique RFID for identification and tracking purposes. An RFID reader is needed to scan the medicine into the smart home system and to detect the

location of the medicine within the smart home. Depending on the user preferences a reminder system can include an alarm, speaking a message or a pop-up message on the TV. Different actuators are needed to perform these tasks and remind the resident to take the medicines. This system has a conflict detection routine which needs to interact with other sensors, actuators and applications. As seen all these task require the interaction of several devices and applications but not with the resident. This data eventually need to be available to the resident, nevertheless the main purpose of the system is the device-to-device interaction and not the user-to-user or user-to-device interaction. Therefore an application like medicine management fit the definition of wireless hive networks.

Food management using a smart refrigerator and a smart microwave is another important service provided by a smart home [25]. In a food management application, RFID tags are used to identify the food items. An RFID reader is used to detect and locate the food. A database with the food information such as name, nutrition facts, and expiration date is needed. Temperature sensors will help to ensure the food remains fresh and at appropriate temperature. Weight sensors are needed to detect when running out of food. A reminder and conflict checking system will also use information from the food management system. In the food management system the food information needs to be available to the user but the interaction among devices is more important. The operations of the sensors, actuators and other devices such as storing information and conflict checking routines needs to be optimized. Again the main focus of the food management service is the device-to-device interaction and not the user-to-device interaction.

These are just two examples of application within a smart home that uses and integrates several sensors and actuators whose interaction need to be optimized for that purpose. Another applications and services in the smart home includes location and tracking, opening and closing the door, wake up service, emergency detection, turning appliances on or off (such as stove or iron) and more. All these applications and services require close interaction among the devices but there is no standard way to do it at this point. Most of these services and applications are integrated in a manual and ad-hoc way. Therefore a set of requirement

for sensors and actuators need to be defined so that the development time and the cost of development can be reduced. The next section lists these requirements.

3.3.5 Smart home requirements

The smart home has a number of devices interacting with each other. There is a need for these devices to be able understand each other. Different sensors use different communication protocols and some protocols are proprietary. The use of wireless mesh networks [60] to allow communication among different protocols would enable devices to communicate with each other. A wireless mesh network uses wireless mesh router which are routers that understands different communication protocols and route from one protocol to another protocol. For example a router that understands Zigbee, Wi-Fi and Ethernet can be used to interconnect three different networks and exchange data among them. This interconnection is necessary and also that the data sent must be correctly understood by other nodes. To this purpose a software middleware layer is presented in which the raw sensor data is sent over the wireless mesh network, the wireless mesh network communicates with the middleware, and the middleware translate the data to a format understandable by the receiving node. For example the Phidgets sensors [72], returns raw data in numbers from 0 to 1000. In the case of a temperature sensor a returned value of 200 is not very useful. The middleware will take care of making the conversion from this raw value to the appropriate temperature in Celsius or Fahrenheit degrees. This middleware also will make easier application development as programmer can just query for temperature without worrying about which physical sensors are being used.

Other requirements especially for battery operated sensors, is that they must have a long life and be reliable. Some perpetual sensors can be plugged into a constant source of power and other can be wireless. Either way sensors should notify with plenty of time when running out of battery or failing. Also nodes should provide security by notifying when they are under attack. Sensor must ensure privacy and security by allowing only the appropriate parties query the reading of the sensors and access them. The architecture for smart home applications is presented in details in the next section.

The need for a solution that is extensible and scalable is needed as some services in the smart home might be temporary. For example imagine the case where the resident of the home is under certain treatment to determine if the resident has certain health condition. It might be possible that the resident needs to carry a set of wireless sensors for a just a few days. These devices will become new to the smart home. Configuration and testing should be minimal and transparent, a process that is not possible in today's smart homes. The resident might not want to alter the smart home configuration and re-program it. Having an easy way to add these sensors to the smart home and remove them when no longer needed, will be very helpful. This requires a common architecture which is presented in the next section

3.3.6 Smart home layered architecture

The following is the proposed architecture for the smart home sensor applications. At the very bottom we have the Phenomena and Objects Layer. In this layer the phenomena to be sensed by the sensors and the objects controlled by the actuators are found. The next layer is the sensor and actuators layer that consist of sensor and actuators from different manufacturers. We assume that these devices implement different protocols. The next layer is the connectivity layer. This layer has the wireless mesh routers which enables communication among different protocol and allow the sensors to communicate among themselves. The next three layers will reside on the smart home host computer. The layer on top of the connectivity layer is the software middleware layer. This software middleware layer will be responsible for implementing the device specific details for each of the sensors and actuators and provide them as services. This will add a layer of abstraction in which instead of reading raw sensor data you will read significant values. These services provided by the software middleware layer will be made available to the service layer. The service layer is a collection of all sensor, actuators, and application services all under the same framework or platform. These services can be incorporated in service-oriented frameworks such as OSGi [4]. This service layer will provide the services to the applications layer. It is in the application layer where the different smart home operations are performed and the one with which the resident interacts with the smart home. In Figure 3.3 you can see a visual representation of this architecture. We have implemented several demos using this service-

oriented approach and using sensors and actuators from different manufacturers in our smart home lab. A description of some of these follows in the next section.

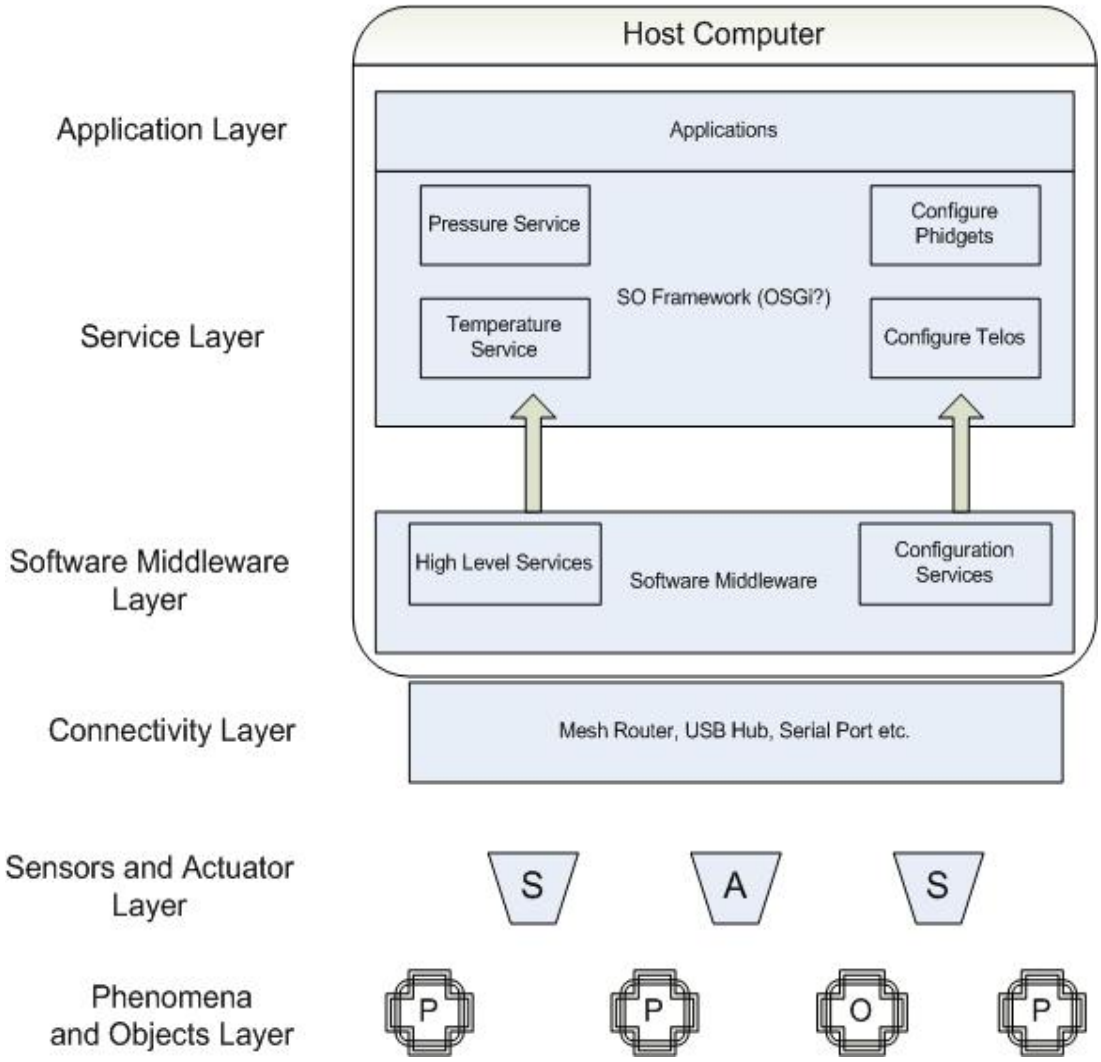

Figure 3 - Smart Home Layered Architecture

3.3.7 Smart home demos

In our smart home lab, we have several demos that use devices such as Phidgets sensors, rfid reader, and web cameras, and X10 appliance controller, and telos motes. Some of these demos interoperate with different technologies by invoking services provided by the service layer. In our case the middleware is also implemented as a service. Applications are therefore developed using only the services provided, abstracting the details on the particular hardware that do the readings. For example we have a temperature service which can either use Phidget sensor or a telos sensor. For the application it does not matter which sensor we use as it uses the temperature service which abstracts the particular hardware that makes the temperature readings. We have implemented several demos using this approach such as the medicine management, a smart fridge, a camera controller and an alarm system that turns on appliances at the time to wake up.

To show interoperability among different protocols we will describe in details one of the demos we have developed. This demo uses wireless sensors to collect the data and communicate with an application via the Internet. We have one telosb mote A which is sensing phenomena. We have another telosb mote B which is a base station node connected to the host computer over a USB port. Node A sends a packet to node B using the protocol IEEE 802.15.4 for Wireless Personal Area Networks. Node B communicates with an OSGi service which reads the packet from the telosb node. The raw data is transformed into meaningful data by the service. This data is then passed to another service which communicates with an application via the Internet. This application consists of a server that receives the data and reads is loudly using a text-to-speech module. Figure 3.4 provides a visual diagram of this demo. The response time between readings the data, send it to the service, send it to the speech-application via the Internet took less than two seconds in all the tests. This is very reasonable time frame for a smart home application where response time is not as critical as industrial applications. Also taking in considerations that the OSGi framework and the sensor network application are running on a VMWare Virtual machine running Xubuntu Linux, all the services are running in an OSGi framework and the speech application is running on a MAC reachable over the Internet. So even after all these layers of

abstractions the response time is very reasonable for smart home type applications. The fact that service oriented approach can cross several platforms in a reasonable amount of time is also a plus.

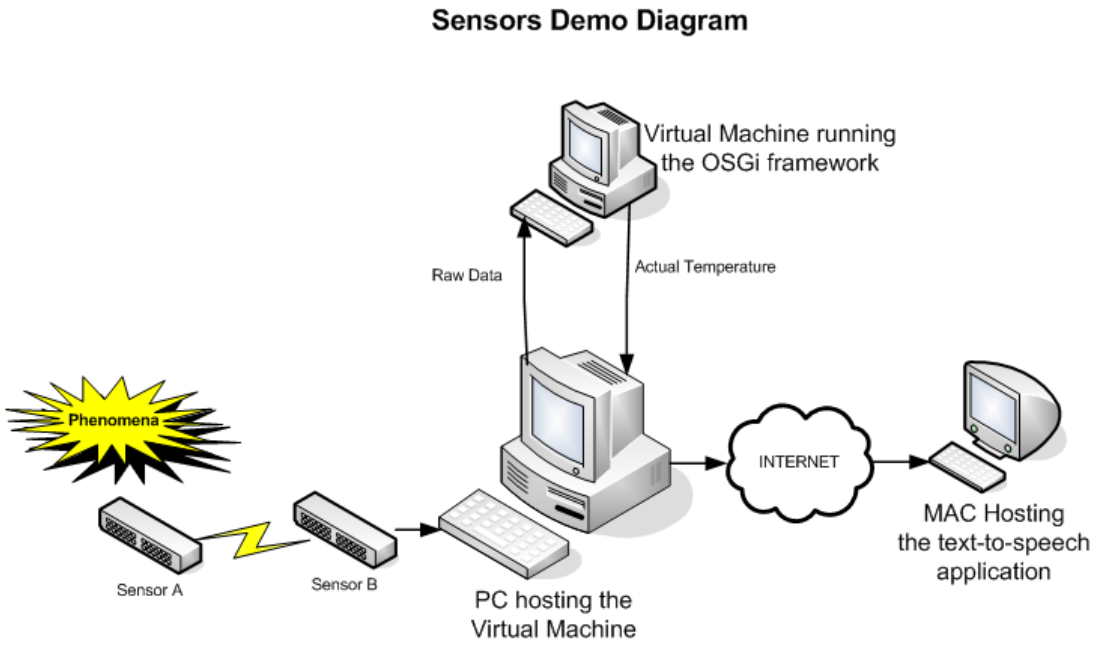

Figure 4 - Sensors Demo Diagram.

3.3.8 Conclusion and future work

A smart home has several sensors and actuators providing different services. Interconnecting them is difficult and is often done in a manual, adhoc way. We identified key requirements for smart home applications that use several sensors and an architecture that will ease the task of development and reduce the costs. The approach focuses on the device-to-device communication and the use of a software-layer middleware. We showed that this approach applies to smart homes in application such as medicine and food management as the tasks the devices need to perform are not designed specifically for human-to-device interaction. The architecture emphasizes a mesh layer which will make communication among different protocols possible and a service-oriented software middleware layer which will make

possible to abstract the devices to the service they provide without worrying on the particular device. We mentioned how several demos on our smart home lab used this approach and gave the details on one of the demos that use different communication protocols and layers of abstractions to show the feasibility of this proposed architecture and middleware. The response time for these applications and demos is acceptable for smart home application. Future work includes refining the middleware layer and tests it with more devices. Making this middleware plug-and-play is another research goal.

3.4 Composition of Services for Notifications in Smart Homes

Modified from a paper published in Proceedings of the Second International Symposium on Universal Communication, 2008

José M. Reyes Álamo, Tanmoy Sarkar, Johnny Wong

3.4.1 Abstract

Giving notifications in a timely manner is an essential service that Smart Home should provide. Communication of resident's data to their doctor, health care provider or family member via email, phone call, or text message is indispensable**.** In order to reduce the cost, improve extendibility and reduce the developer's burden and learning curve, we propose a service-oriented notification system. This system has different simple services which provide essential communication needs, and by composing these services with applications more sophisticated needs can also be satisfied. A detailed description and the architecture are elaborated below.

3.4.2 Introduction

The challenges of holding mobility make it difficult to live alone independently for elderly adults. Smart home environments provide assistance to elderly ones and persons with special needs in performing their activities of daily living. Smart Home aims at providing a more independent life to those people who require only a little assistance, so that they can stay at

home instead of moving to some nursing facility. The emerging wireless technology and pervasive computing model help us to create prevalent environments to support the aged people.

The objective is to design a cost effective, easy to use notification service influenced by contextual factors [73], which means the notification system decides the best time and form for presenting messages, depending on the state of its users and their environment along with estimating the value of the message content. This notification system uses a service-oriented architecture. The architecture will create independent services for different notification needs and will compose these services among themselves, with other services and applications to create more advanced notifications.

3.4.3 Related Work

Considerable research is going on medical alert system [74] and medical monitoring for independent living [75]. The emergency response systems like [74] and Home alert system like [75],[76] guarantee a reliable solution for our daily life but the initial investment and the recurrence cost to choose these services may be expensive for middle-class people. These solutions are also fixed for a particular set of devices and configuration of the system. Future Smart Home environments will contain a wide variety of devices and services from different manufacturers and developers. Also these devices and services are expected to be added and removed in a dynamic way. In this context, Service Oriented Architecture is the solution which will help us to achieve the vendor independence platform and architecture. Service Oriented Architecture separates process into distinct services which can be distributed over a network. The processes can be combined and reused as per application requirement. Our application meets these requirements minimizing any deployment or recurrence cost.

3.4.4 Our Approach

Smart Home is an application of pervasive computing that integrates devices and services into everyday living environment to benefit their inhabitants. A complete smart home architecture will have the following components: device network with different smart home appliances, vendor independent service-hosting platform and a service centre for managing

networks and service deploy functions [77]. Open Service Gateway Initiative (OSGI) tries to meet these requirements by providing a managed, extensible framework to connect various devices in a local network. The general smart home design is shown in Figure 3.5 [78].

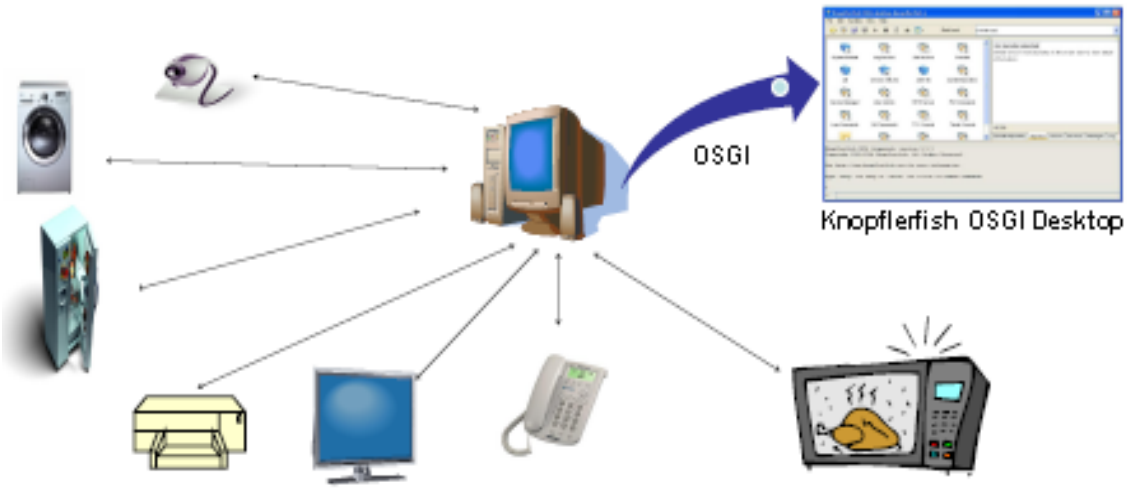

Figure 5 - General Smart home design

In a Smart Home environment there is a continuous need for communication with external entities in different ways. Sending data, email, text message, or making automated phone calls are a constant necessity. Usually it is up to the programmer to learn how to use and program these functionalities in order to incorporate them in the applications. This is tightly-coupled to the particular programming language and platform the developer use. Therefore the developer has to learn all the details and get the required libraries in order to provide these notifications in different applications and configure them with the particular details of each user.

Our idea is to provide these different means of communication a message as services within the Smart Home. This way an application can consume a service instead of

implementing it. These services will be personalized for the person who lives in the Smart Home. Data such as the email, phone number and address need not to be entered or queried in each application. This will allow the developer to focus on other aspects of development and not on notifications details. We understand that this will be very beneficial in terms of cost, development time and learning curve.

Notifications are used very frequently by a lot of applications and modules. These programs will definitely reduce the time and cost of development. Also learning the particular details to provide certain notification can be time consuming. Moreover, if new technology arrives it will be hard to integrate with existing application and a re-implementation might be the only way to put them together. The developer will have to learn again the details of the technology and it definitely will increase the cost and time of development. By providing these ways of notification as services, they can be easily used and updated without affecting the applications using it, as they will follow a service-oriented approach. Also these services can be easily composed to create more sophisticated services by combining them with other services and applications within the Smart Home.

3.4.5 Architecture

In this section we will present a detailed architecture of our approach. We also describe a set of demos we have developed in our Smart Home Lab that make use of this approach and the benefits we have received of using notifications as services. Figure 3.6 shows this basic design architecture of the smart home notification system developed by our research group [24].

In our Smart Home Lab we use phidgets, telos motes, Insteon, X10 and various other types of intelligent sensors [72]. As shown in Figure 3.6, Home Network applications can be subdivided into different components such as Home Control, Security, Health and Entertainment. These applications have been developed as OSGi bundles which have been deployed as OSGi services [8],[79]. These application services are running in an OSGi framework. This allows OSGi services to provide other applications and services with the required notification functionalities. Our Notification bundle is also a service deployed in the

OSGi framework. This bundle is being used by all the other applications shown in Figure 3.6 such as Home Control, Security, Health and Entertainment. We will briefly describe a few of the demos developed using this approach below

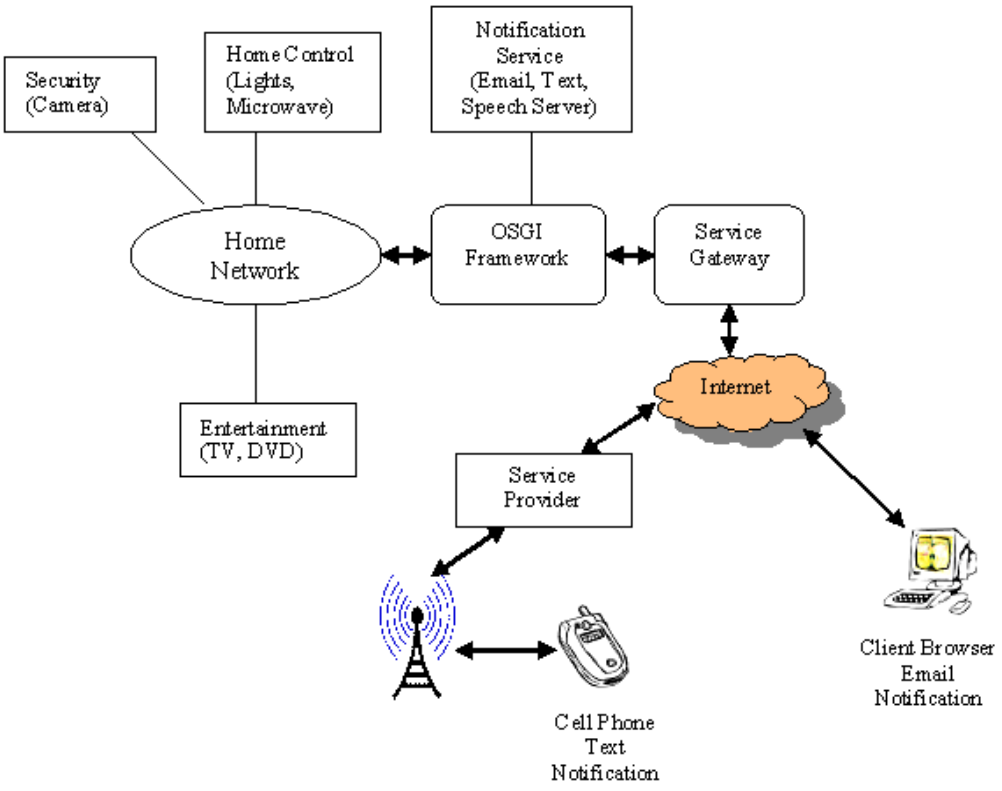

Figure 6 - Basic Design Architecture

Demos:

Microwave demo:

In this demo a microwave oven is controlled using the computer and an RFID reader. RFID tags are used to identify the food or meal. When a tag is read, the food data and cooking information is queried. The microwave uses this information to automatically set the appropriate time and temperature for the food preparation. The

notification services are used to say loud what food is being cooked. The notification system also notify when the food is ready. It also displays an appropriate message. The reminder message can be played by the computer using notification speech server and displayed on the TV.

Medicine demo:

This demo helps in the management of the medicines in the Smart Home environment [68]. This demo integrates the doctor, the pharmacy and the smart home to ensure safety and check for conflicts. Prescriptions are checked for conflicts with other prescriptions, with over-the-counter medicines and conflicts with food. This demo also reminds the elder people with special needs, of critical tasks such as medication intake and doctor appointments. These reminders use the speech notification service. Also notifications can be sent to the health care provider in the case that the person misses a dosage of a medicine. When a conflict arises also a notification can be set to the doctor or health care provider and also to the resident. These notifications can be customized and messages can be sent to end user's mobile phone or to email account depending on the preferences. Also if the person is not within the home network and cannot hear the speech notification, a message alerting of this situation can be sent also. This shows how many different uses can be given to this notification services. The developer does not need to create them but just use them.

Smart Fridge Demo:

This demo helps in tracking food items in the Smart Home**.** Food is identified by RFID tags. A particular food item has some data indicating the resident's preferences of items and threshold weights. The preference order will indicate if the resident wants the food to be automatically reordered and the threshold weight will indicate when it is a good time to reorder. The reordering can be done by sending a notification to the grocery store when the weight goes below the threshold for a particular food item. This demo can also inform end users using the notification service when important food items such as milk and eggs goes below the threshold

value to the shop nearby. The notification service is also used in conjunction with the medicine demo to detect conflicts between food and medicines. When a conflict arises a message can be sent to the appropriate parties and the resident notified. All these services can be used by the developer but there is no need to re-develop them.

Macro Demo:

Powerful technology does not always have user-friendly nature. As a consequence, usability of technology reduces. As an example, clumsy look of Knopflerfish OSGI Desktop may make users reluctant of using OSGI applications. Secondly Centralized complex applications may not become useful for elderly ones and people with special needs. The goal of the Macro demo is to Control or schedule any Smart Home applications on the fly and improve the quality of life of elderly people by using distributed architecture with wireless technologies and the Internet. In this demo we have developed a platform independent graphical user interface called Application Management System by which an end user can check or schedule device applications connected in home network. Notification Service may be used as a component of this demo. The example given below will clearly demonstrate the usefulness of Notification Service in Macro Demo. Suppose an old person is at home alone and is unfit to walk. The son is at the office during daytime. It may be possible that the person accidentally falls from the sofa or bed and his son wants immediate notification when an accident like this happens so that he can help his father. Therefore, he has scheduled an event using the macro demo that whenever the pressure sensor attached to his father's wrist triggers, a text notification will be sent to his mobile phone. Our notification text message service easily fits into this scenario without modifying a single line of code and perform the operation expeditiously.

The above examples clearly demonstrate the potential benefits of having notification technologies in Smart Home environment to support independent living. The service-oriented approach allow to reduce the deployment and development cost as the developer needs to learn only what the services do and how to use and does not have to develop the services

every time. This also shows how the same service can be used several times by different applications. So a repetitive task now does not have to be repeatedly developed; only repeatedly used.

3.4.6 Conclusions and Future Work

A service-oriented approach significantly simplifies the task of satisfying simple communication need as well as more complex ones. We presented the advantages of such approach and also demonstrated how smart home applications can benefit from it. Reducing the cost and time of development is essential for smart homes to become ubiquitous. Having these basic functionalities available as services will greatly reduce the learning curve of the developer as well as the customization that each smart home requires. As our prototyped implementation shows these simple and combined services markedly simplify the development of applications and satisfy the different needs for communication via various means.

A major challenge in future notification system will be to predict immanent importunity of messages. Additional user studies are needed to capture this and to create prediction models. Although our proposed notification service has an audio based service, there are several new trends in Smart Home that can be integrated to our notification service such as video based multimedia service. By using our notification service through ubiquitous computing the dream of future smart home will come true.

3.5 Using Web Services for Medication Management in a Smart Home Environment

Modified from a paper published in Proceedings of the 7th International Conference on Smart Homes and Health Telematics 2009

José M. Reyes Álamo, Johnny Wong, Ryan Babbitt, Hen-I Yang, Carl Chang

3.5.1 Abstract.

The Smart Home is a house equipped with technology to assist especially the elderly and persons with special needs. Smart Homes rely on Service-Oriented technology usually OSGi. Web Services (WS) receives little emphasis on Smart Homes, but they can be very useful for some applications. That is the case of management of medication as this task can become very difficult and involve different, remote parties. Several solutions have been proposed for applications like medications management but their lack of interoperability limits them. We present a solution that integrates current systems and provides interoperability by using WS. The secure transfer of sensitive data among subsystems is achieved by using secure WS for communication purposes as shown by our prototyped implementation.

3.5.2 Introduction

The Smart Home (SH) is a house equipped with technology like sensors and actuators with the purpose to help the resident in performing their activities of daily living. SH research has focused on using these homes to help the elderly and person with special needs to stay home longer and live more independently. SHs have relied on the Service Oriented Computing (SOC) approach to simplify its design, shorten the development time and reduce the cost [18]. Different Service Oriented Architectures (SOA) has been developed. Two widely used SOA are Web Services (WS) and OSGi services. Both architectures are platform independent, rely on well defined standards and can be deployed over networks. WS has become very popular and most research in SOC has focused on WS. OSGi has become a widely used standard for applications that use embedded devices such as SHs. This section shows that WS are very useful for certain SH applications especially remote applications and those involving third parties. The use of WS for certain applications certainly reduce the cost and development time as some of these services needed are already available.

One of these applications of WS within the SH is with the management of medications. Difficult prescription names, different instructions and the fact that a person might be taking several medicines at the same time can make this a difficult task especially for the elderly and persons with special needs. Using technology will certainly help this population to increase compliance and medication intake [80].

The SH environment can help with the management of medications. The authors in [66] provide a system called Medicine Information Support System (MISS) which integrates the doctor, the pharmacy and the SH to increase safety, manage the medications and increase medication intake and compliance. This system requires little action from the SH resident as it takes care of the process in a transparent way. WS technologies are integrated with the MISS system to make it more interoperable, expandable and platform and language independent.

3.5.3 Related Work

In [81] the author explains how today's devices are more powerful with more computational and communication power allowing this technology to be used in home applications. However these devices and appliances use different protocols, a lot of them proprietary. They proposed a solution based on WS to achieve interoperability, heterogeneity and scalability. They implement the WS stack in the devices or in the devices controller. Their main purpose for bringing technology into the home is to help the elderly to stay home longer, having the necessary assistance and reduce the learning curve, sharing a similar motivation of this work.

Even though in [81] the authors propose a solution based on WS for SH applications, in general SHs rely on the OSGi service platform [8]. Both technologies offer a set of unique features, so the combination of them can help providing new solutions. In [82] the authors describe a driver on top of OSGi in which devices can be dynamically added, invoked and mapped to different WS standards. Their vision is to have a centric platform in which the functionalities are provided as services and applications compose these services in a flexible way. The OSGi platform is used and drivers on top of OSGi are developed to support WS. Each WS is mapped into a service in the platform. This is a good attempt in trying to combine both technologies together but full integration it is still needed [67].

3.5.4 OSGi and Web Services in the MISS System

The Medicine Information Support System (MISS) is a system that helps patients to manage their prescriptions [66]. It improves safety by checking for conflicts among medications, health conditions and food. The way it does it is by integrating the doctor, the pharmacy and

the SH in such a way that prescription information is forwarded from one subsystem to the other. It uses a trusted third party that defines the conflicts among medications, health conditions and foods. Prescriptions data is checked against the patient's data stored at each subsystem to identify any possible conflicts. The next few paragraphs briefly summarize MISS and how we applied WS to it.

The doctor subsystem is where this process begins. The doctor enters the prescription details with the medications and dosage information. The doctor is assumed to have a record of the previous prescriptions and health conditions of the patient. The patient's record is checked against the new prescription to determine if a conflict exists. To detect a conflict we use a trusted third-party such as the Food and Drug Administration (FDA) or the Physician Desk Reference (PDR) who defines conflicts among medications, conditions and foods. After this check, prescription data is forwarded to the patient's preferred pharmacy via a secure communication channel.

The pharmacy subsystem performs a similar function of checking the prescription for conflicts with other medications. We assume that the pharmacy keeps a record of all medications previously picked up by the patient. A check for prescription conflicts is performed using the patient's data at the pharmacy and a trusted third party such as the FDA or PDR. If no conflicts are found a secure channel is used to forward the prescription data to the SH.

The SH subsystem performs a similar check for conflicts. It is assumed that the SH keeps an inventory of the medications and the food at home. The SH then check for conflicts among medications picked-up from different pharmacies and with food at home. Again a trusted third party defines the different conflicts.

The MISS system assumes a secure communication channel for forwarding data from the doctor to the pharmacy and from the pharmacy to the SH but no details of this secure channel are provided. One way this secure communication channel can be implemented is by using WS which will also make the different subsystems interoperable as WS are platform and

language independent, a main feature of this technology [29]. This will allow for current system to keep working but also to be expanded by using this technology.

This work extends the MISS system by making WS a fundamental part of it. MISS makes use of stand-alone applications for the doctor and the pharmacy subsystems. It uses OSGi services at the SH to control the devices and application. This work uses WS to glue these three subsystems together providing interoperability and preserving the platform and language independence. It also allows other applications to use the Web Service provided if needed. This approach takes care of providing a secure mechanism for forwarding data among subsystems. The next section has a summary of a prototyped implementation of MISS using WS.

3.5.5 Formal Verification

In applications using Web Services that involve thirds parties, maintaining the privacy of the data is an important issue. We want to make sure that the system is safe, that the privacy is respected and that the system is correctly used. These issues are addressed by using formal methods. One part is to check that the conflict checking routine is sound and that it detects the conflicts it is supposed to detect. Another part is to make sure the privacy of the patient's data is respected when using the Web Services or another service-oriented approach. Another part is to ensure that the person is using the system correctly. The next subsections provide more details how to achieve these goals.

Model checking is a mathematically well-founded automated software verification technique that is becoming popular as software becomes more complex and intractable, as a way to ensure safety and correctness [83]. In model checking a model M of the system is provided with state transitions and a property P. The checker will visit all possible paths of state transitions to see if P holds. If the property holds it will return true, otherwise it provides a trace file indicating where the property fails to hold. In the MISS system [66] a model for the conflict checking routine is provided which defines a set of medicines M, a set of medical conditions C, a set of food items F, a set of doctors D, a set of pharmacies P, and the smart home H with functions for detecting conflicting_medicines: $M \rightarrow P(M)$,

conflicting_conditions: M→ P(C) and conflicting_food: M→ P(F). Given a prescribed medicine m, these three functions are computed and the results are compared to the patient record. If the medicines on the patient record and the conflicting medicines form a disjoint set, then no conflict is found otherwise, the prescribed medicine creates a conflict and the system gives a notification. In MISS an instantiation of the model is given which shows that the routine correctly detects a determined conflict. Nevertheless this is no proof that all possible conflicts are detected. Using model checking techniques we have a stronger argument in favor of the model used for the conflict checking routine. In this work that is precisely what is done, to ensure that the prescription data is formally checked and a broader amount of conflicts are detected and not just providing one example. We make use of the SPIN model checker [84] to perform this task which for the medicines, conditions and food the model checker detects those conflicts.

Service Registry	Policy Store	Context Manager	Privacy Manager
-stores service descriptions and binding information -returns this information to the Privacy Manager when requested	-stores the privacy policies of the system -returns a set of rules applicable to a given request when queried by the Privacy Manager	-retrieves context data from appropriate services and supplies it to the Privacy Manager	-receives service invocation requests -coordinate other components to evaluate a request -returns the access decision

Figure 7 – Major components of our policy architecture

We propose a privacy model based on the service-oriented computing paradigm, where services are software artifacts that control zero or more devices, interact with each other via invocations, and may be remotely monitored or operated [85]. The components of this privacy model are Service Registry, Policy Store, Context Manager and Privacy Manager. A summary of these components can be found in Figure 3.7. Different entities might have different policies and the context will vary. This privacy model ensures that the privacy of

the patient's data is respected. Any set of policies imposed by any of the subsystems will also be enforced. Only authorized parties will have access to the data. This is a necessary step to comply with state and federal laws as well as to respect the desires of the patient, doctor and pharmacy on how to handle the sensitive data. The interactions among these components can be seen in Figure 3.8. It can be noticed that after a service is invoked, in this case Web Service, this privacy model will look for the service in the registry and then it will retrieve the applicable policies for this service, user etc. It will consider the context and based on the analysis access to the service and data will be granted or denied.

Technology can improve compliance with medication intake [80]. Still it is needed to ensure that patients are using the system as expected by taking the right medicine with the right dosage at the right time. Two main requirements to ensure medication intake are medication on-time and medication completeness. For example if 4 pills a day need to be taken every 6 hours, we want to ensure that the patient follows those instructions. The patient fail the on-time requirement if takes the pill every 3 hours or every 5 hours for example. The patient fails the completeness requirement if less than 4 pills or more that 4 pills are taken during the day. In order to check for this we formally check that the medicine detection mechanism detects when the wrong medicine is chosen, that the notifications and reminders systems [68] give appropriate reminder and warnings in case of an error, that the context is taken into consideration [46] and the system in general correctly catches situations in which the on-time and completeness properties are violated.

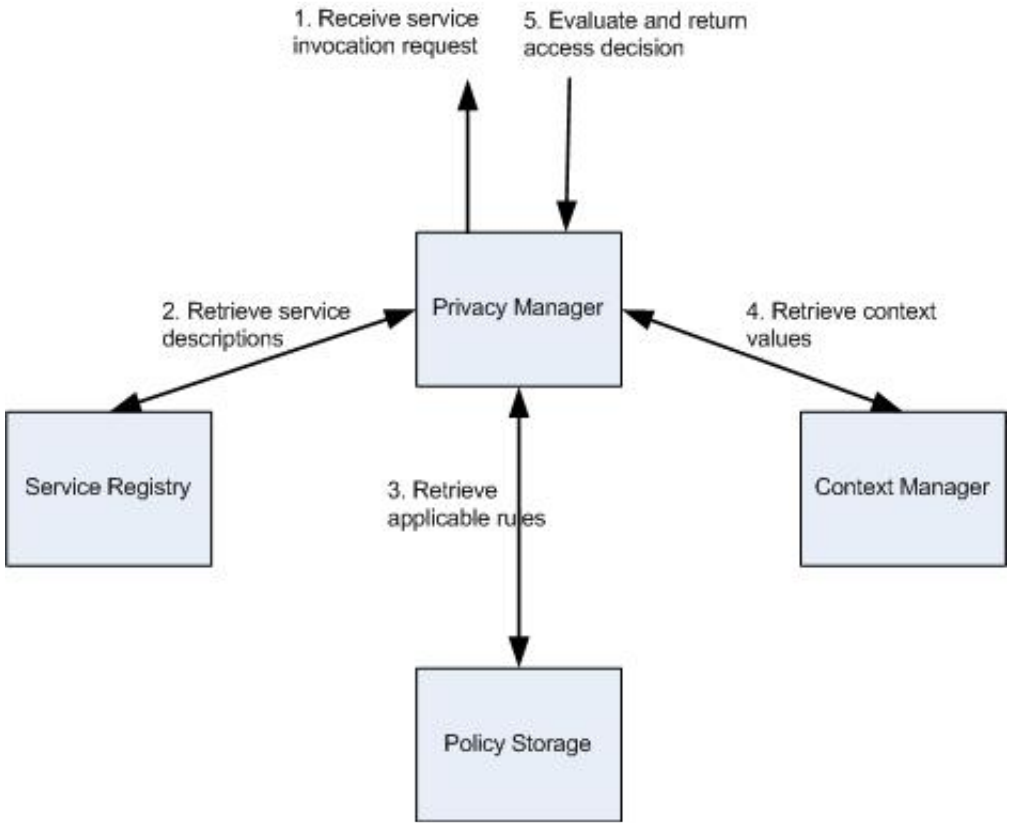

Figure 8 - Request Evaluation Process

3.5.6 Prototyped Implementation Using Web Services

A prototyped implementation using the approach described above has been implemented in our SH Lab. For the doctor's subsystem an application has been developed in which the prescription details are entered and then forwarded to the pharmacy using a secure WS provided by the pharmacy that the doctor's subsystem invokes. The pharmacy WS provide the tools for the doctor to forward the prescription's data. When a prescription's data is received via this WS, the data will be transferred to the pharmacy's main system so that the pharmacist can starts preparing the prescription. When the prescription is ready the pharmacy subsystem use a WS provided by the SH to forward the prescription data from the pharmacy into the home system. This WS will allow data to be transferred from the doctor's subsystem

to the SH also, such as medical conditions. This allows the SH to have more data available and perform and final and more complete check for conflicts which will improve safety.

3.5.7 Conclusion and Future Work

The SH is a house equipped with technology to help the elderly and person with special needs to stay home longer. One way to help this population is with the management of medications. This is a task that can become difficult given the amount of data that needs to be handled. To increase compliance and medication intake several solutions have been proposed but the lack of a universal platform and language independent approach limit these. A solution that integrates current systems and makes use of service oriented approaches, specifically WS, is presented. Future work includes the use of formal methods to validate that the data is transferred correctly, that privacy is respected and that the entire system performs its intended functionality. Also full integration among SOA such as WS and OSGi and the use of some WS sub-languages is something we would like to do in the future.

CHAPTER 4. SUPPORT FOR MEDICATION SAFETY AND COMPLIANCE IN SMART HOME ENVIRONMENTS

Modified from a paper published in the International Journal of Advanced Pervasive and Ubiquitous Computing, IGI Global 2009.

José M. Reyes Álamo, Hen-I Yang, Ryan Babbitt, Johnny Wong, Carl Chang

4.1 Abstract

The rapid pace of new medications introduced to the market and the trend of modern healthcare towards specialization complicates doctors' prescribing process and patients' management of medications as well as increases the likelihood that a patient may be prescribed an unsafe medication. The severity of this problem is magnified when patients require multiple medications or have cognitive impairments. The Medicine Information Support System (MISS) is designed to integrate related information systems from doctor offices, pharmacies and patient's smart homes with a universal database of medication conflicts to enable safety checks for adverse reactions among prescribed medications. MISS enhances the quality of patients' healthcare by supporting the conformance of patients' medication intake. MISS is also designed to ensure patients' medical records remain private by following the privacy guidelines and regulations such as the Health Information Portability and Accountability Act (HIPAA) law in the United States.

4.2 Introduction

A Smart Home (SH) integrates and networks different technologies to provide assistance with activities of daily living. SHs specifically designed for the elderly and persons with special needs have gained importance as a research subject in the last few years as the baby boomer generation reaches the retirement age and begins to experience the need for assistance [1]. One of the primary needs of this population is assistance in managing medications, which can be challenging due to complicated medication names, multiple simultaneous medications, or medications with differing types of dosing instructions. Two

often overlooked but extremely important facets of medication management are the detection of possible conflicts among medications and complying with prescriptions.

The Medicine Information Support System (MISS) presents a smart home-based solution to integrating doctor offices, pharmacies and a patient's home to assist patients in managing their medications [66]. It supports the safety of prescriptions by checking at multiple times for conflicts between medications, health conditions and foods that a patient may be eating. When a doctor enters the information about a new prescription, the system automatically checks for conflicts with any previously prescribed medications or diagnosed health conditions in the patient's local record. The prescription information is then forwarded to the pharmacy to determine if there is a conflict with any prescriptions filled at that pharmacy, including prescriptions filled from other doctors. Lastly, the prescription information is then forwarded to the SH to check for conflicts between all of the patient's medications, health conditions, and foods, regardless of doctor or pharmacy. This allows for medications prescribed by different doctors of filled at different pharmacies to be checked for conflicts.

MISS is designed to transparently wrap around existing computer systems that are already in place in doctors' offices, pharmacies and smart homes using Web Services (WS) to provide interoperability via standard communication interfaces. The system requirements and design architecture are presented in section 4 provides a detailed view of the interactions between different subsystems, as well as the data they exchange. Because proper medication management is critical to patients' well being, a medication management system must be error free and capable of detecting various types of medication conflicts. A model is presented in section 5 to demonstrate the correctness of MISS. Additionally, since prescriptions and health conditions are personal health information, the storage, use, and disclosure of this information between systems presents potential threats to patient privacy. As a result, privacy regulations such as HIPAA [86] must be incorporated into the requirements and design of MISS to provide proper privacy protection for patients'.

The rest of this chapter is organized as follows: the requirements, design, and detailed architecture of MISS are presented after a discussion of related work. We then describe a

system model for conflict checking and compliance monitoring and illustrate MISS's prototype implementation. The last section summarizes the contributions of MISS and states future work.

4.3 Related Work

Some previous efforts have been made for helping individuals manage their prescriptions. For example, the Magic Medicine Cabinet (MMC) [87] is an Internet-enabled medication manager equipped with facial recognition software, RFID smart labels, and vital signs monitor and voice synthesis. The MMC gives personalized reminders, detects when a wrong medication is taken, and measures vital signs. However, no details are provided on the authors' claim that their system interacts with the patient's pharmacy, doctors and health care providers, and no safety checks are made for conflicts among medications. Our work bridges this gap by using WS to connect the patient's SH with the patient's doctor and pharmacy and to conduct multiple checks for medication conflicts.

The Smart Medicine Cabinet (SMC) [88], and the Smart Box [89],[90] extend the Magic Medicine Cabinet by using passive RFID tags to identify medication containers and Bluetooth technology to synchronize the MMC with a patient's cellular phone. The SMC is automatically updated when the cell phone is brought within range. The major drawback of these two systems is that the patient must remember to carry the cell phone to the pharmacy as well as near the MMC. Our system presents a more user-friendly solution, especially for those with cognitive impairments, in that it does not require special user interaction or a cell phone.

Technology for automatic dispensing of pills also exists [91],[92], typically, manual configuration is needed to set notification and dispensation times. Also, the medications also have to be manually removed after a reminder is generated.

All of these solutions certainly facilitate the task of taking medications, by assisting with prescription compliance [89],[93], giving reminders to the patient [94],[95], and detecting missed doses. However, they do not provide a networked solution and still require significant manual input, which would be difficult for patients with cognitive disabilities. This limitation

is addressed in [96] where a system for patients with dementia is presented. Our system makes a similar of related technology to help patients with prescription compliance, but our work is more comprehensive as it transparently networks the smart home with existing doctor offices and pharmacies but does not require the patients to enter any data.

In addition to these systems, several software applications have been developed to assist with the medication prescribing process and the management of patients' medical records. Such applications often target mobile devices such as cell phones, smart phones, and PDAs [97]. Nevertheless, they are intended for people with these devices and familiar–do not provide support for safety checks for conflicting medications or assistance in monitoring compliance with a prescription. For patients' record keeping, solutions, such as Google Health [98] and Microsoft Health Vault [99], are available to warehouse the patient's health and medications records. However, it remains to be seen if patients will be comfortable storing personal information in remote enterprise environments. Pervasive spaces like SHs can be used to assist patients in performing activities of daily living (ADL) [18], but also can help patients with the management of their medications [66],[80],[1]. We would like to have a system that can integrate all these independent solutions in order to have a stronger management of medications.

4.4 System Requirements and Design

A successful medication management system must have a set of well defined requirement and a well planned design. The requirements and design of MISS are discussed below.

4.4.1 System Requirements

Describing the processes of how a person receives a prescription and takes a medication reveals several important system requirements.

Obtaining a prescription involves the following steps:

1. The patient visits the doctor.

2. The doctor prescribed medications.

3. The patient visits the pharmacy and gets the medications.

4. The patient at home intakes the medications.

Medication intake involves the following steps:

4.1 Wait for the next dosage time

4.2 Locate the medication container

4.3 Open the container

4.4 Extract the appropriate amount of medication

4.5 Intake the medication

4.6 Close the prescription container

4.7 Return the container to the medication cabinet

The aforementioned related work can help to automate some of these steps. For instance, the Smart Box or Smart Cabinet can automate steps 4.1 and 4.2 and automatic pill dispensers address steps 4.3 and 4.4. When patients with cognitive impairment “waits” for the next dosage time, they might forget when the next dosage time is [96], hence a system that automatically sets up the dosing schedule using the prescription’s data, generates the reminders and notifies if a dosage is missed will help to improve the compliance with medication intake [80]. SH can also improve the task of locating the medications container, by incorporating unique identifiers for the prescription containers like RFID tags as well as a system that can detect these containers in the entire space [100].

Because manual entering of prescription information by the resident would likely be unreliable due to inexperience and human error, the prescription information should be

entered by an expert and automatically forwarded into the SH system without repeated manual entries at various locations such as pharmacies or the home. Thus, in MISS, prescription information is automatically forwarded between doctors, pharmacies, and homes. The next section presents the system design and operation of MISS in greater detail.

4.4.2 MISS Design

MISS consists of three main subsystems: the Doctor Subsystem (DS), the Pharmacy Subsystem (PS) and the Smart Home Subsystem (SS) which are operated by four main actors: the doctor, the pharmacist, the patient, and the SH, respectively. Each system is assumed to have a local database that stores information about the patient/inhabitation, and a global medication conflict database (MCD) exists that defines all conflicts between medications, conditions, and foods. The MCD is maintained by a knowledgeable and trusted third party. Figure 4.1 illustrates the interactions between these actors and subsystems.

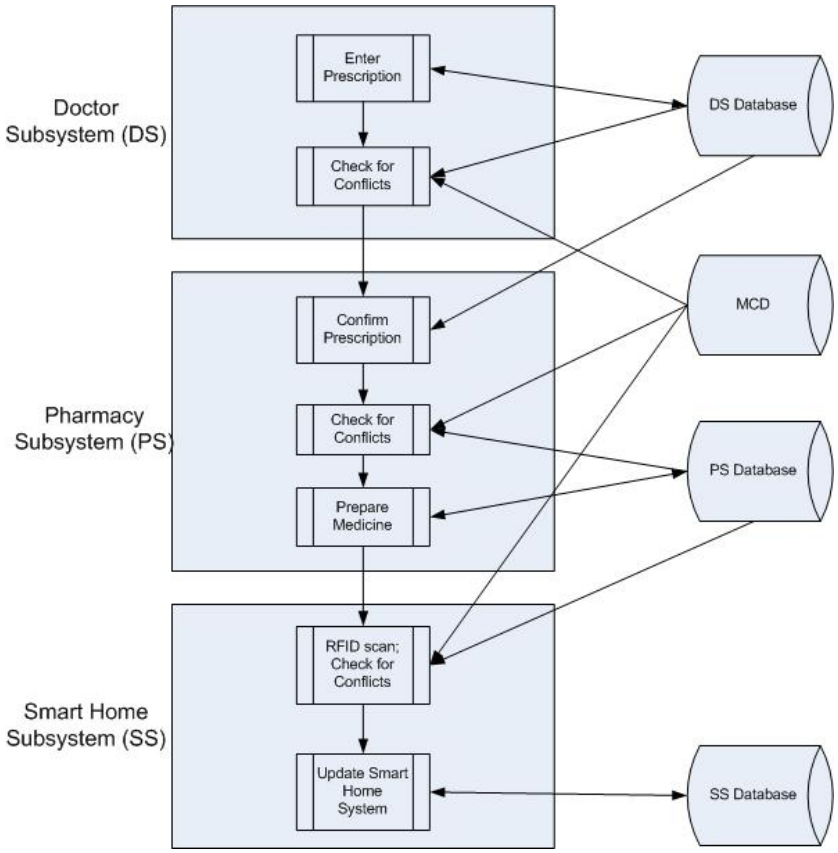

Figure 9 - Medicine information support system diagram

The more important use cases and actors for MISS are shown in Figure 4.2. The Update Smart Home action will be responsible of updating any subsystem that might use the medication data such as medication inventory and notifications.

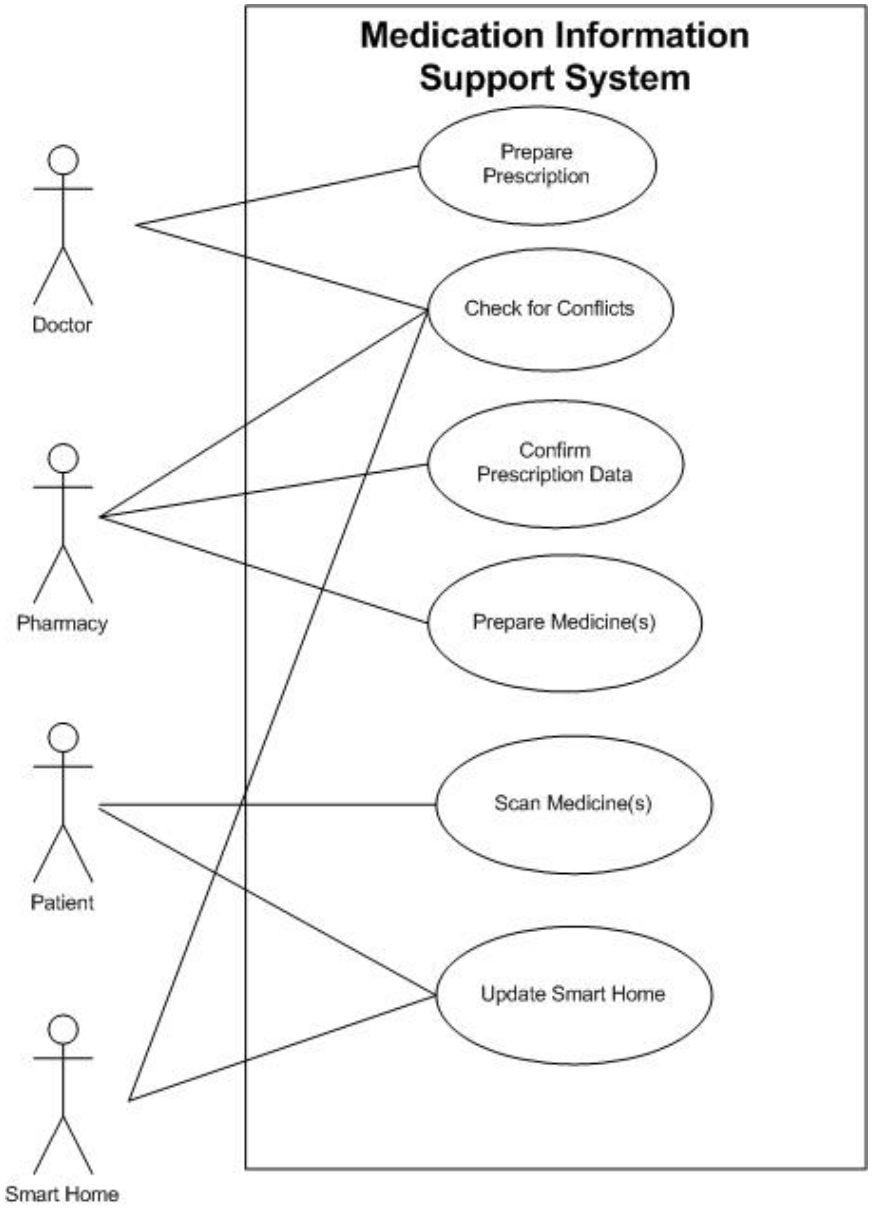

Figure 10 - Medicine information support system use cases and actors

4.4.3 Doctor's Subsystem (DS)

The process starts when the doctor prescribes a medication. MISS can assist with the prescribing process by the conflicts among medications, information that the doctor can use as a reference. The patient's medical record of previous prescriptions, the current health conditions, and the symptoms and patient's preferences are used to determine possible conflicts and reactions. This way the doctor can prescribe medications that do not create any conflict or allergic reactions. The lists of medications, side effects and possible conflicts can be retrieved from the database maintained by a trusted third party such as the Food and Drug Administration (FDA) [101], or the Physician's Desk Reference (PDR) [102]. A conflict-free prescription will be forwarded to the pharmacy either immediately to the patient's preferred pharmacy or after a request from the pharmacy that the patient will choose later.

If the data is forwarded to a preferred pharmacy, it is assumed that the doctor subsystem and the pharmacy subsystem share a unique ID that will be used to identify the patient. Using the ID, the doctor subsystem will check its local database for conflicts with the new prescription. If no conflict is found, the prescription's data is forwarded to the patient's preferred pharmacy through a secure channel. The doctor's office will also issue a customized printed-RFID prescription which will be used later by the pharmacy module.

If the patient chooses a pharmacy later, the process is almost the same. The main difference is when the data arrives to the pharmacy. In the first case it arrives immediately in the second case data arrives when the patient gets into the pharmacy and the pharmacist request the prescription information from the doctor's office. Now let us consider PS design in more detail.

4.4.4 The Pharmacy Subsystem (PS)

There are two possible starting scenarios for the pharmacy subsystem: When the patient chooses the pharmacy at the doctor's office or when the patient chooses the pharmacy later.

Consider the first scenario, when the patient chose a preferred pharmacy at the doctor's office. The pharmacy receives the prescription's data over a secure channel when the patient is still at the doctor's office, allowing the pharmacist to start preparing the prescription before

the patient arrives. The pharmacy issues the prescription in special RFID-enabled containers that allow the system to uniquely identify the medication in them. The prescription is checked for conflicts with other medications that have been recently filled for the patient, similar to the check performed at the doctor's module. However, this additional check is necessary as the patient might be receiving prescriptions from different doctors but picking medications from the same pharmacy.

In the second scenario, when the patient has chosen the pharmacy manually, the patient arrives to the pharmacy with the printed-RFID prescription. This prescription is given to the pharmacist as if it is a regular prescription, but this time the prescription is ready or in process as the information was received previously. This printed-RFID prescription is scanned and the data is compared with the data forwarded by the doctor's office. If no conflict is found with the new prescription, the medication is dispatched into an RFID-enabled container; otherwise, the pharmacist is notified and doctor are notified.

Our system can reduce the waiting time at the pharmacy if the prescription data is forwarded immediately from the doctor's office allowing the pharmacist to start preparing the prescription while the patient is on the way. This is an excellent feature for the elderly population who might need their medications as soon as possible or want to avoid long waits or several trips to the pharmacy. Another benefit of our system is the extra layer of safety by double checking for conflicting medications using the pharmacy's local data.

4.4.5 The Smart Home Subsystem (SS)

When the patient arrives to the SH, its local database is updated by scanning the prescription container with an RFID reader. If the range on the RFID reader is large enough, this requires no manual effort as the RFID tags will be read once the patient is within their range [103]. This will retrieve the prescription's data from the pharmacy using a secure communication channel. After the data is received, a final check for conflicts between the new prescription and current medications, health conditions, and foods in the home will take place. This third level of checking is necessary as the patient might be receiving different prescriptions that are filled from different pharmacies. Also, food information is available only at home,

requiring this check to be done at the home. If a medication conflict or health condition conflict is found, the appropriate parties are notified. Otherwise, the medications are placed in a SMC or loaded into an automatic medication dispenser and the patient notified of any potential food conflicts.

In addition to checking for the safety of a new prescription, we want the patients to comply with the doctor's instructions by taking a proper dosage at the proper time. To evaluate patient's compliance with a given prescription, there are two factors to consider: timeliness and completeness. Timeliness determines whether the medications are taken at a correct time, and completeness defines whether the correct dosage is taken. Formal definitions of the timeliness and completeness are presented later with the system model, which is followed with a description of how to use the context information [46] and the notification services [68] in a smart home to aid prescription compliance. The next subsection contains details on how interoperability among subsystems is provided.

4.4.6 Interoperability by Using Web Services

MISS is built on top of stand-alone applications already running the operations for doctors and pharmacies therefore interoperability among these subsystems is needed. The prescription information is sensitive data which will be shared among doctors or pharmacies only if the patient authorizes it [86]. A secure communication mechanism among subsystems to transfer this data becomes necessary. WS can provide this secure communication as well as interoperability as WS are platform and language independent and network friendly and secure [29]. Each subsystem will provide a secure WS to allow communication with other subsystems in an efficient and secure way.

4.5 Designing for Privacy

Because MISS stores, uses, and discloses personal health information, it is also mandatory that related privacy legislation such as the Health Information Portability and Accountability Act (HIPAA) [86] governs its design and operation. To demonstrate how this is the case, we identify the problematic aspects of MISS from a privacy perspective and show case by case

that our design is in line with the corresponding HIPAA requirements. There are essentially four high-level problem areas (PA) that HIPAA addresses:

PA1. The access and use of Protected Health Information (PHI) by employees within a health system or health information database

PA2. The disclosure of PHI to other systems and health service providers

PA3. The ability of the individual to access and amend PHI

PA4. The ability of the individual (or a governing body) to audit the uses and disclosure of PHI including the management of individual consent.

Since MISS essentially wraps web services around existing doctor, pharmacy, and SH systems and integrates them with a global database, it is the disclosure of PHI among these systems (PA2) that is of primary concern. Implementing proper access control policies and other data security safeguards (PA1) are the responsibilities of the individual doctor and pharmacy systems and not of the MISS framework proper. If these systems correctly provide these mechanisms, then we can assume that only authenticated and authorized doctors and pharmacists can access PHI in their respective systems. Similarly, it is the responsibility of the original systems to provide access, amendment, and audit mechanisms that individuals can use to check, correct, and challenge the uses and disclosures of their PHI (PA3 and PA4). If these mechanisms are provided, then MISS can also wrap them with WS in a way that preserves the functionality of the original mechanisms but provides the interoperability and universal access of WS. The cases in which health information is sent between subsystems are 1) the transfer of information from doctor to pharmacy, 2) from doctor to home, and 3) from three systems to the conflict detection service.

Case 1: Doctor Subsystem Disclosure to Pharmacy Subsystem

HIPAA permits the disclosure of PHI to other health care services providers if they have a previously established relationship with the individual and the disclosed information is

necessary to that relationship. This rule applies to the transfer of prescription information from a doctor system to a pharmacy system as an identifier is transferred along with the prescription information. However, since the individual has chosen the pharmacy to which the identifier and prescription are routed, it follows that the necessary relationship between pharmacy and individual must exist.

Case 2: Doctor/Pharmacy Subsystem Disclosure to the Smart Home Subsystem

The second case in which PHI is disclosed is from either the doctor or pharmacy subsystem to the home subsystem following the diagnosis of a new condition. However, because the home subsystem is viewed as a trusted representative for the individual, it follows that the doctor subsystem should be permitted to disclose this information to the home. Moreover, this information does not contain personal identifiers. On the other hand, since the doctor subsystem is communicating with an individual home subsystem (i.e. the home of a particular patient), the communication could be reasonably used to identify the individual if the destination of the messages were known. Therefore, secure communication is required to prevent information from leaking to eavesdroppers. The incorporation of secure WS in the MISS framework satisfies such requirement.

Case 3: Doctor/Pharmacy/Smart Home Subsystem to Conflict Detection Service

The fact that the doctor, pharmacy, and home subsystems submit medical information to the global conflict detection service also pertains to privacy. However, because both the doctor and pharmacy submit completely de-identified health information (only medication and health conditions) for conflict detection and de-identified information is returned by this service, there is no privacy risk inherent to these communications. On the other hand, the invocation of the conflict detection service by the home system does pose a privacy threat according to HIPAA. The fact that the request is coming from a system operating on behalf of an individual could allow the owners of the conflict service to identify on whose behalf the service request was made, thus linking the medication or condition to the individual. The MISS framework mitigates this situation with two security mechanisms. First, encrypted communication is done between the home system and conflict detection service, preventing

eavesdropping. Secondly, a trusted anonymizing proxy is used by the home system to aggregate web traffic from multiple sources, sufficiently decreasing the likelihood that such information can be attributed to the original individual.

4.6 System Architecture

The successful operation of MISS requires proper collaboration between three independent subsystems: DS, PS and the SS. Each subsystem maintains separate local databases as well as expected functionalities required by doctors, pharmacists and patients. Information exchange is triggered when certain events occur that can cause conflicts among medications; however, only the necessary information is exchanged. In addition, it is assumed that there exists an external MCD containing information regarding conflicting medications, health conditions and food, which is maintained by trusted third parties such as FDA or PDR.

MISS is intended to be an extension of existing DS and PS, not a replacement, and one of our primary goals is to maximize the interoperability of these existing systems. Since most of doctors' offices and pharmacies already have some kind of computer systems to perform their daily functions and their respective employees are comfortable and accustomed to using these systems, it is essential that MISS works with different existing systems. Our system architecture achieves this by encompassing existing systems with wrappers, and introduces WS as the universal interface for communication between subsystems.

Since a patient can visit multiple doctors and multiple pharmacies, MISS employs WS as the mean of inter-subsystem communications, which shields the details of internal implementation of each subsystem from other subsystems as long as they offer compatible WS.

The architecture of MISS can be best presented in two tiers. The higher tier deals with the inter-subsystem communication that mimics the actual actions and interactions among patients, doctors and pharmacists. The lower tier describes the detailed structural make-up and inner operations within each subsystem.

The higher tier overview (illustrated in Figure 4.3), shows that MISS includes three subsystems, DS, PS and SS which utilize the external MCD system, assumed to be maintained by a trusted third party. Each subsystem provides a human computer interface that allows interactions with user in certain role, and a system interface implemented as a WS that allows communications between subsystems.

The primary owner and user of DS (illustrated in Figure 4.4) is the doctor who, after each examination, enters diagnoses and prescriptions into the system as needed. Before a prescription is finalized, MISS checks for medication conflicts by anonymously sending over the patient's prescribed drugs and medical conditions from DS to MCD. If no conflicts with the new prescription are found, the prescription information is then stored into the local database, and forwarded to PS; otherwise a notification is given to the doctor, along with suggestion on alternatives.

Once the new prescription information arrives at the pharmacy (illustrated in Figure 4.5), it automatically triggers the retrieval of other current prescriptions associated with the patient, and a check for conflicts is sent from PS to MCD. This request makes it possible to identify potential conflicts between medications prescribed by different doctors, who may not be fully aware of other drugs the patient is currently taking. Once the new prescription is cleared of conflicts, the pharmacist would prepare the drugs, while PS would send back a confirmation to DS and forward the prescription to SS. If a conflict is found, a notification is given to the pharmacist who might contact the doctor or take other appropriate measures.

The SS is assumed to maintain two sets of data, the patient's health information and the food inventory of the home. When the prescription is dispatched, PS would send over the prescription information to SS. After the patient picks up the prescription and returns home, the medication needs to be checked into the automatic dispenser. MISS then retrieves the information about the medication during this process and performs a final check for conflicts, including checking for food conflicts with the new prescription. Any discrepancy or conflict will notify the patient, the doctor and the pharmacist. Figures 4.5-4.8 describes the interactions within the SH subsystem.

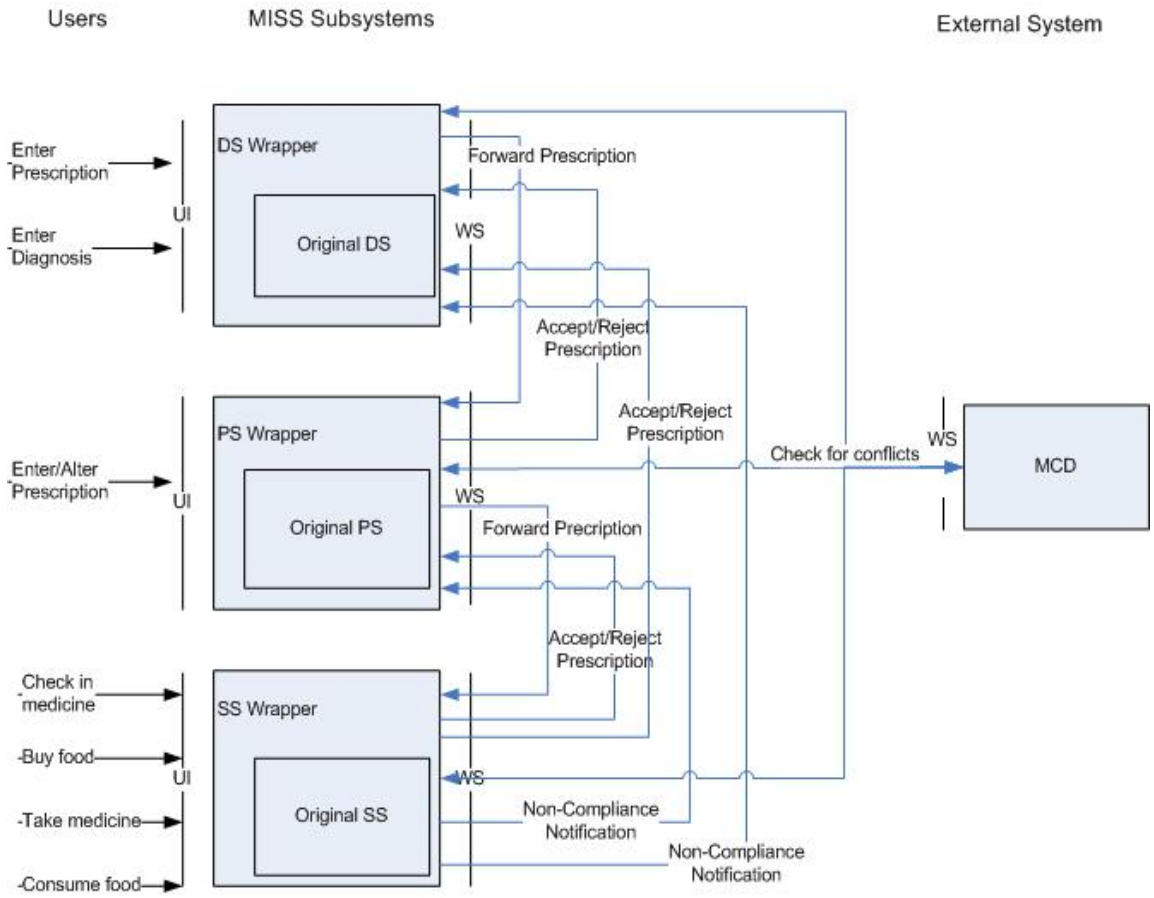

Figure 11 - Communications between MISS and MCD

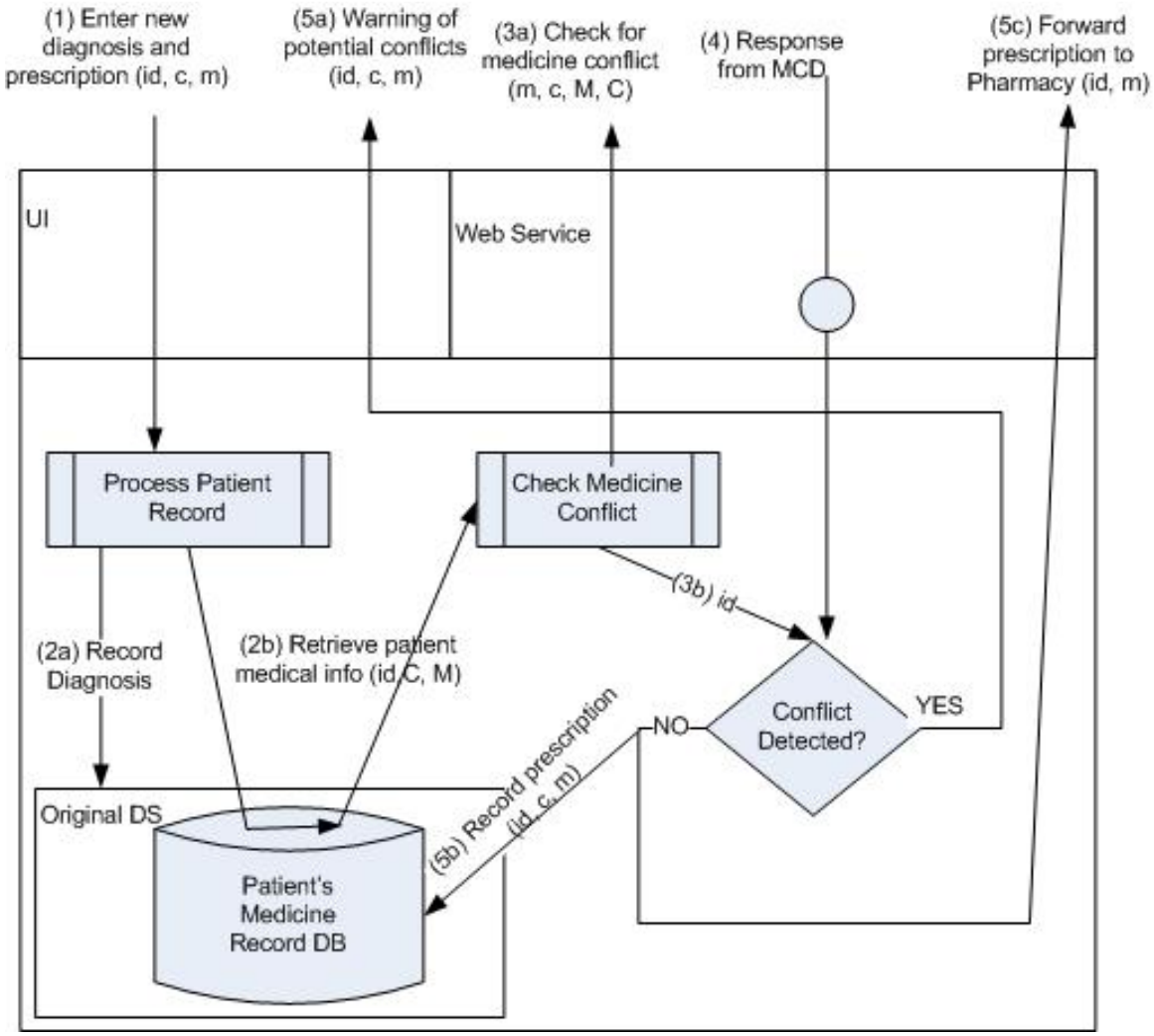

Figure 12 - Doctor Subsystem

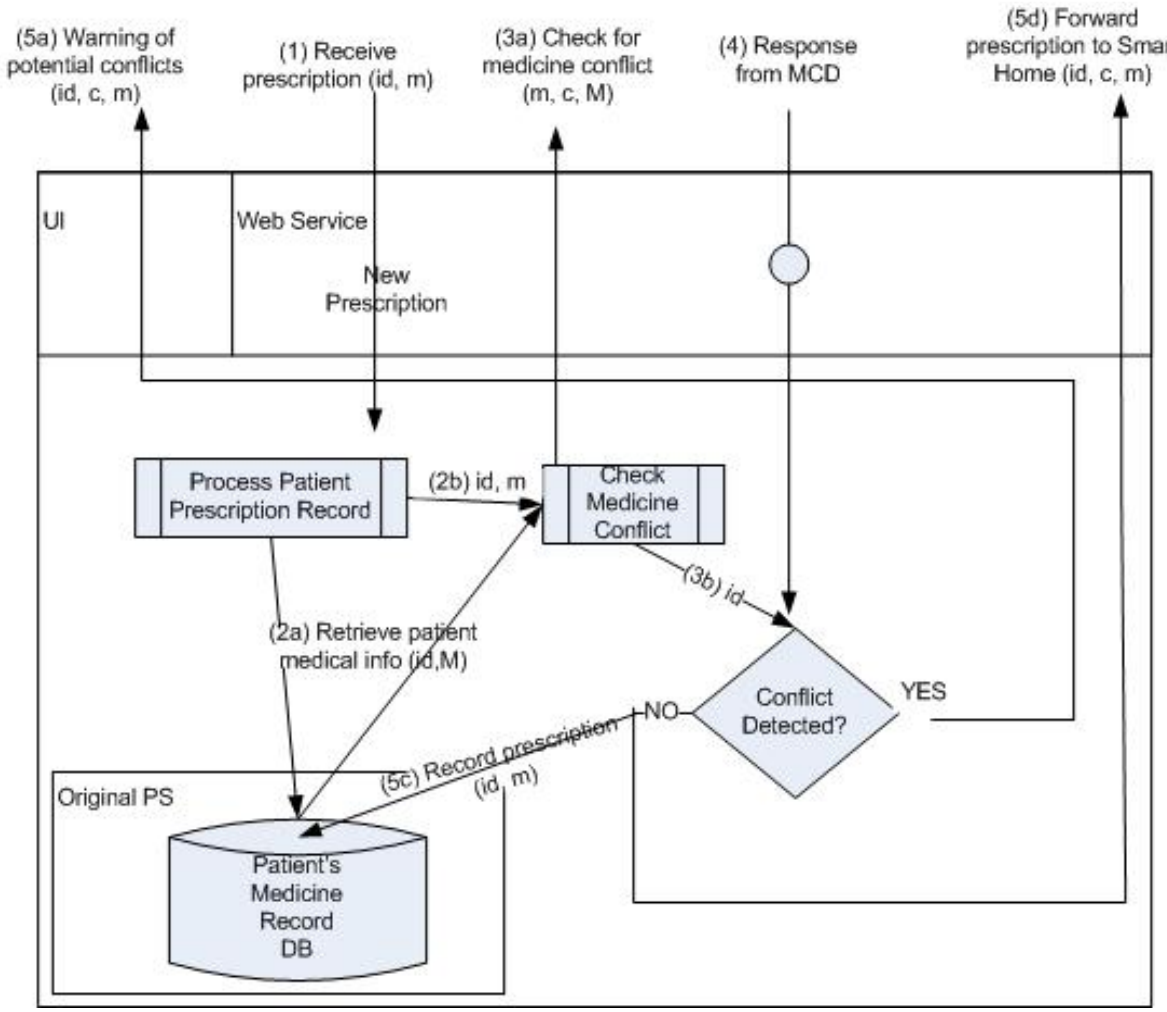

Figure 13 - Pharmacy subsystem

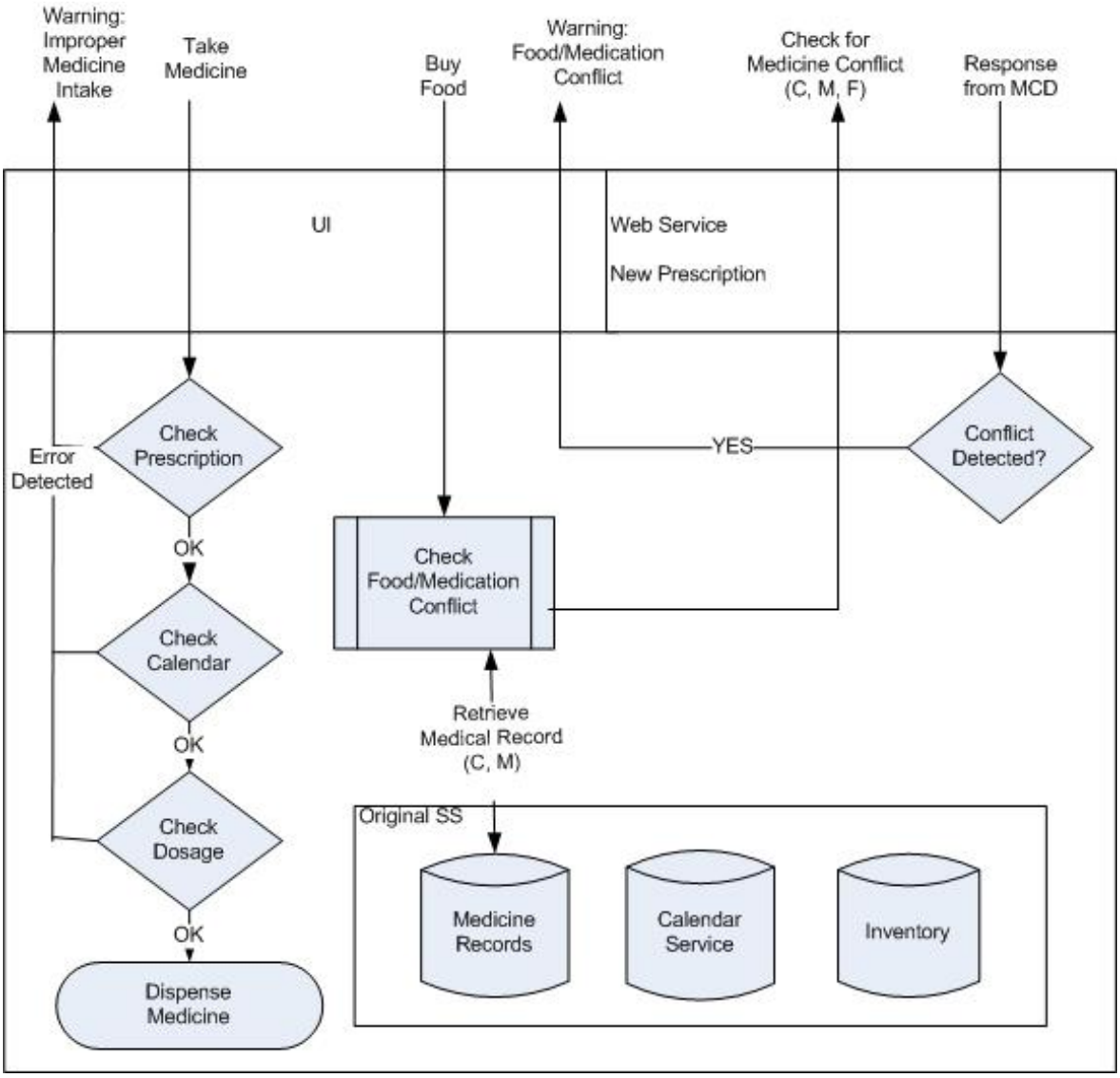

Figure 14 - Smart home subsystem

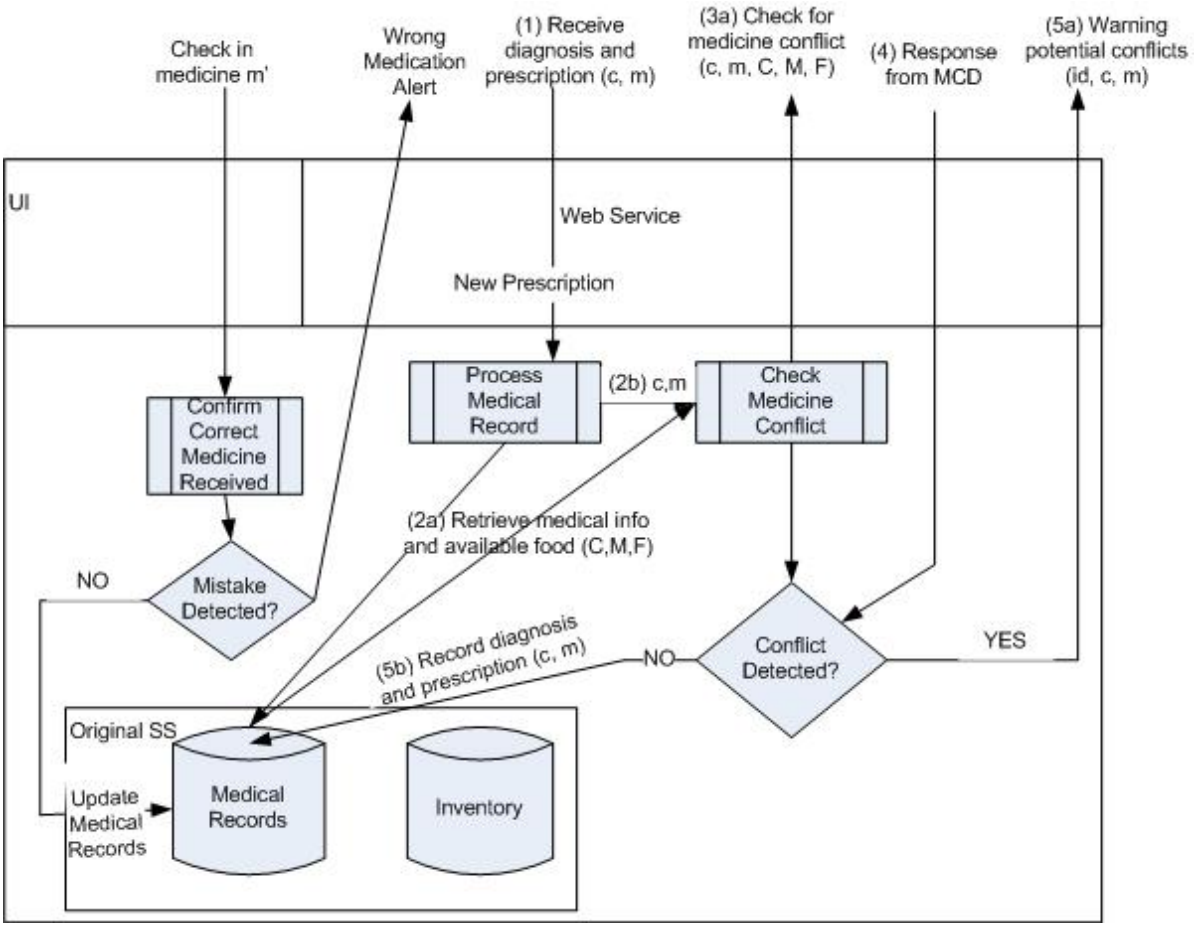

Figure 15 - Smart home subsystem 2

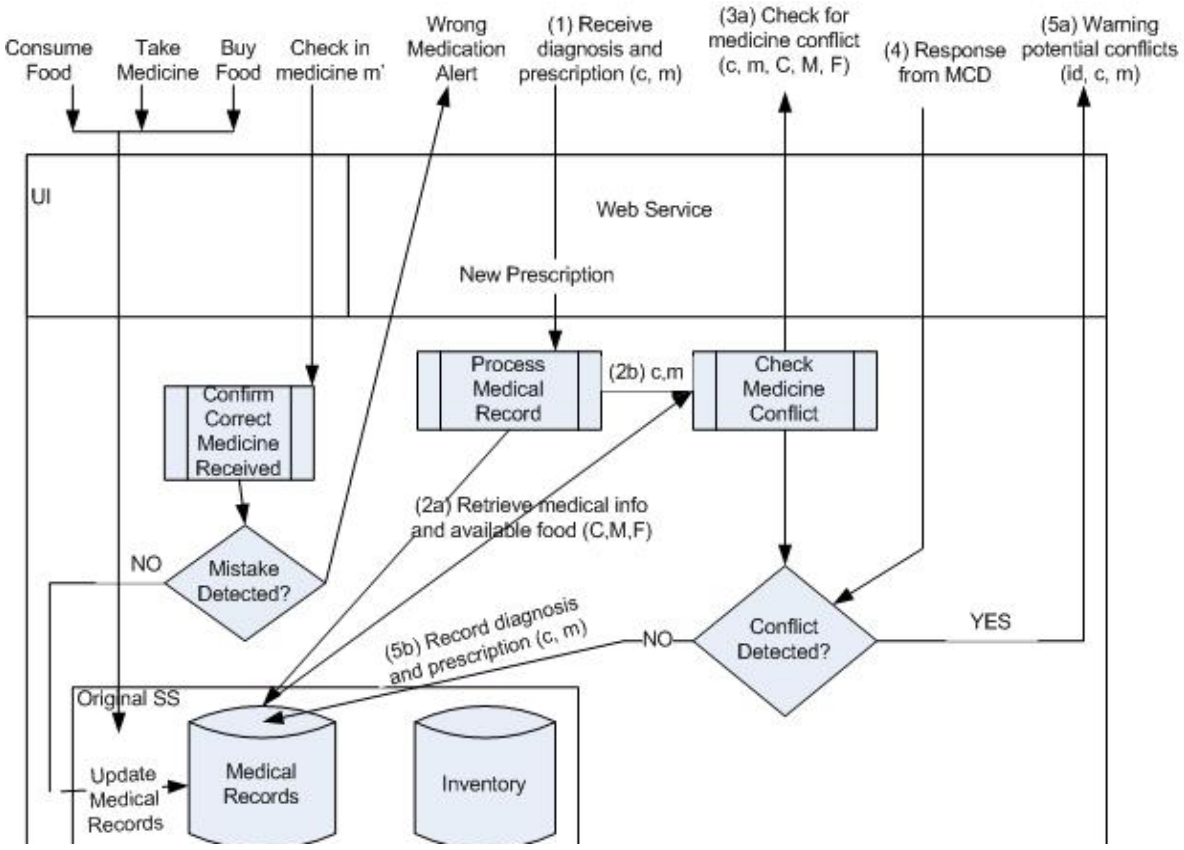

Figure 16 - Smart home subsystem 3

4.7 System Modeling

This section contains a model of the medication management system to further define and examine conflict detection and compliance with different types of prescriptions. Model details for checking for conflicts and checking for compliance by fulfillment of the timeliness and completeness requirements are in the next sections.

4.7.1 Model the MCD

A medication management system must be accurate, reliable and provide safety by detecting conflicts that a new prescription may cause by interacting with other medications, health conditions, or foods. The trusted MCD defines the conflicts over the set of medications *M*, a set of food *F*, and the set of health conditions *C* using several functions over the set of

medications. The function *conflicting_medications: M→P(M)*, where *P(X)* denotes the power set of *X*, returns the set of medications that conflict with a given medication, the function *conflicting_conditions: M→P(C)* returns the set of health conditions that conflict with a given medication, and the function *conflicting_food: M→P(F)* returns the set of foods that conflict with a given medication. We define the Medication System Model as follows:

Definition

Our Medication System Model consists of the following sets:

- *M*, set of medications
- *C*, set of medical conditions
- *F*, set of food
- *D*, doctors, hospitals or clinics the patient visits
- *P*, set of pharmacies where the patient gets prescriptions
- *H*, the patient's Smart Home

With the following functions:

- *conflicting_medications: M→P(M)*
- *conflicting_conditions: M→ P(C)*
- *conflicting_food: M→ P(F)*

A patient *p* is represented by a tuple *p = (id, Mp, Cp, Fp, CMp, CFp, CCp)*. The *id* field uniquely identifies the patient, and $Mp \subseteq M$, $Cp \subseteq C$, $Fp \subseteq F$ be the subset of medications currently prescribed to patient *p*, the health conditions diagnosed to patient *p*, and the foods

patient *p* has available in the smart house, respectively. The sets of medicines, conditions, and foods that are unsafe for patient *p* can be defined as follows:

- $CMp = \bigcup_{m \in Mp} conflicting\_medicines(m)$ represent the set of medications that conflicts with the medications prescribed to patient *p*,
- $CCp = \bigcup_{m \in Mp} conflicting\_conditions(m)$ be the set of medical conditions that conflicts with the patient's medication.
- $CFp = \bigcup_{m \in Mp} conflicting\_food(m)$ be the set of food that conflicts with the patient medications.

These sets will allow doctors or caregiver can make better informed decisions when prescribing medications for a new or existing health condition. Given a newly prescribed medication *m* to patient *p*, the MCD determines what conflicts with *m* by applying the three conflict functions to *m*. If we let the sets of potential conflicts of new medication *m* be $CM = conflicting\_medicines(m)$, $CC = conflicting\_conditions(m)$, and $CF = conflicting_{food(m)}$, respectively, *m* can be determined to be a safe medication if

1. $CM \cap Mp = \emptyset$ and $m \cap CMp = \emptyset$ - None of *p*'s current medications conflict with the new medication and the new medication does not conflict with any of p's current medications and
2. $CC \cap Cp = \emptyset$, - The new medication does not conflict with any of the conditions that patient *p* has) and
3. $CF \cap Fp = \emptyset$ - The new medication does not conflict with any of patient *p*'s foods

If all three checks succeed, then we update the patient's set of current medications as well as the patient's conflict medications, conditions, and foods accordingly

4.7.2 Conflict Checking

Each MISS subsystem, DS, PS, or SS, checks for conflicts with almost the same algorithm, but it is expected that each one can capture a different set of conflicts because of differences

in the local databases. A generalized version of the conflict checking algorithm and supporting functions are defined as follows:

Definition: Get Data (GD)

Input: Prescription *r = (p, m)*

//Get patient's *p* data from the local database

Query *Mp, Cp, Fp, CMp, CCp, CFp*

//Get medication *m* conflicting data from the global database

Compute CM, CF, CC

Definition: Conflict Checking (CC)

Input: Prescription *r = (p, m)*

Call Get Data (r)

If ($CM \cap Mp = \emptyset$ and $m \cap CMp = \emptyset$)

If ($CC \cap Cp = \emptyset$)

If ($CF \cap Fp = \emptyset$)

//No Conflict found

$CMp = CMp \cup CM$

$CCp = CCp \cup CC$

$CFp = CFp \cup CF$

Else

Medication *m* creates food conflict

Else

Medication *m* creates a health condition conflict

Else

Medication *m* creates a medications conflict

4.7.3 Conflicts at Doctor, Pharmacy, and Smart Home Subsystem

In this model the DS provides a new prescription *r = (p, m)* that *r* contains the id of the patient *p* and the id of medication *m*. The DS has a local database with patient's *p* record. The DS does not to store information related to food, therefore the checking at DS focuses on identifying conflicts between medications and health conditions. To perform this checking, we invoke the function *CC* with input *r*. If no conflict is found then the prescription *r* is forwarded to the PS.

At the *PS*, it is assumed that the prescription *r* has already been checked and cleared at the *DS*. The *PS* matches the new prescription r with the patient's local record of previous prescriptions and over-the-counter medications to check for possible conflicts. This second check is necessary as the patient might have prescriptions filled from multiple doctors. Over-the-counter medications may also be obtained at pharmacies and may also cause conflicts. The checking at the PS is performed by calling function *CC* with input *r* using the pharmacy's dataset. If no conflict is detected, the prescription r is cleared and forwarded to the SS.

Once the *SS* receives the prescriptions *r SS* performs a final check for conflict using its local inventory of medications, conditions, and food. This check is necessary as the patient might be having prescriptions from different doctors filled at different pharmacies. These remaining types of conflicts are the ones that the *SS* will be responsible of detecting. Figure 4.9 shows and examples of these multiple paths of conflicts. It illustrates how a patient might be visiting different doctor's and pharmacies in which case some conflicts might go

undetected, but the SH can detect them. It also illustrates how conflicts can be detected early in the process as each subsystem performs a safety check.

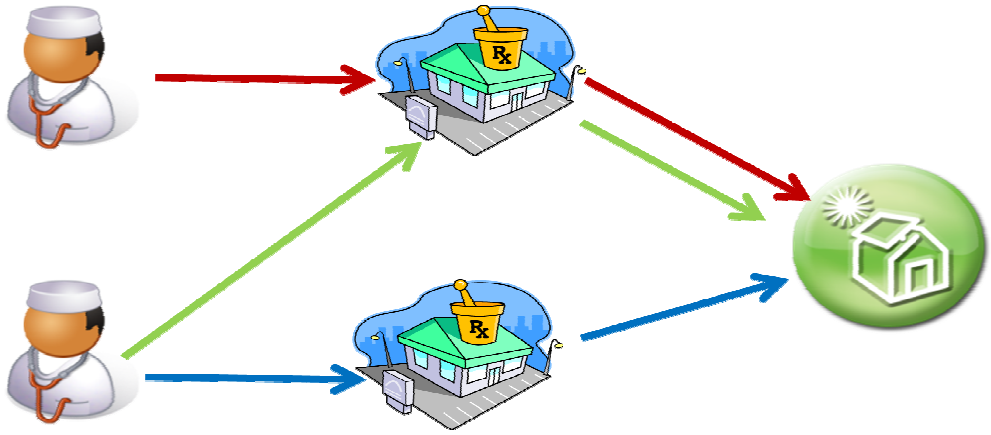

Figure 17 - Multiple paths of conflicts

4.8 Compliance Monitoring

There are two aspects to determining if a patient is complying with a given prescription: if the right medication is being taken in the right dose at the right time [104]. We call the former requirement *completeness* and the latter as *timeliness*. For completeness monitoring, we currently make the assumption that the smart home has the capability to determine if a correct dose of medication was taken. This is not an unreasonable assumption if automatic dispensers are used to distribute the correct dose of medication and patients can be trusted to take all of the medication given them. On the other hand, timeliness proves to be a much more complicated issue.

Different medications often come with different types of intake instructions requiring a medication management system to keep track of which medicines should be taken and when. For this chapter, we study the three common categories of prescription instructions [105] listed below:

1. Do not take more than dosage *d* within time interval *t*

2. Take dosage *d* every time interval *t*

3. Take dosage *d* before (or after) activity *a*

In the following discussions on timeliness, it is assumed that the system has the capability to detect when medications are taken and deduce related context information about the patient, such as if the patient has recently had a meal or is in the process of going to bed. It is also assumed that the system is able to dispense medications and is able to notify patients and doctors based on prescription instructions and a patient's intake actions.

For the first category of prescription instructions ("do not take more than dosage *d* within time interval *t*"), we need to keep track of the amount and time of medication intake. To monitor this condition, we use a common concept of a sliding window to create a *Prescription Sliding Window (PSW)* that continuously adds the amount of the given medication taken in a given time period as follows. When a patient intakes a medication, the system generates an intake event *e* that consists of the medication taken *m*, the dosage d_e of *m*, and a timestamp t_e. Each time an intake event occurs, MISS looks into the *PSW*, which covers the events from t_e-*t* to t_e. If the accumulated dosage $\sum d_{e'} | \ \forall e'$ such that $t_e\text{-}t \leq t_{e'} < t_e$ exceeds permitted dosage *d*, an overdose exception x_{od} is generated and the appropriate healthcare professional notified. In the actual implementation, the SS subsystem of MISS keeps a separate PSW queue for each medication , and when an intake event occurs, is inserted to the corresponding queue and the accumulated dosage updated. The queues are maintained in a lazy fashion, meaning that MISS does not proactively remove intake events prior to $t_{e'}$ - *t* but only purges events when a new intake event occurs and the time period of the PSW does not include them (i.e. they are outdated).

For the second category of prescription instructions ("take dosage *d* every time interval *t*"), the patient is required to space out the medication intake by a certain interval of time. These intervals are setup when the patient intakes the medication for the first time, which we label t_0. Using t_0 as the starting point, the subsequent intake events t_n should occur at $t_n = t_0 + nt$ for n=1 to the size of the prescription. However, instead of requiring the patient is to take

the dose at *exactly* some time t_n, we allow a patient to take a dose at *approximately* time t_n, providing a little bit of flexibility for the patient. To accommodate this flexibility, we introduce the concept of timing tolerance. We define *tolerance tol* as, given a scheduled time t_e for administering a dosage, the patient should take the medication at within some time interval which deviates from t_e by at most *tol*. In other words, only medications taken within the time interval $[t_e - tol, t_e + tol]$ would satisfy this timeliness requirement. Any medication intake outside of $[t_n - tol, t_n + tol]$ would be considered a violation to the prescription instruction. For most medications, this tolerance is in the order of minutes. However, should the patient miss a medication intake in a given tolerance time interval, the patient and the doctor are notified. The system can then adjust the schedule of medication, that is establish a new set of time intervals, based on a newtime $t_{0'}$ when the patient takes the first dosage after missing the previous dose. When the patient takes the medication, the system monitors if the dosage d_{tn} at time t_n corresponds to the prescribed dosage *d*. Figure 4.10 shows a visual representation of these intervals.

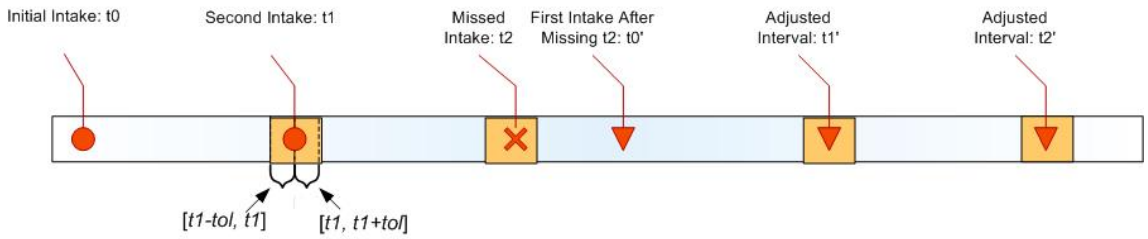

Figure 18 - Timeliness intervals: take dosage d every time interval t

For the third category of prescriptions ("take dosage *d* before (or after) activity *a*"), the scheduled time for a patient to take the medication is relative to a certain activity that the patient performs, such as eating a meal or preparing for bed. We will use the same notation as in the previous discussion, assuming the patient takes the medication at time t_e, and the timing tolerance is *tol*. If the prescription instruction dictates "taking medication with dosage *d* before activity *a*," the system checks back at $t_e + tol$ to see if activity *a* has occurred since

t_e. On the other hand, if the medication is supposed to be taken after an activity *a*, the system first check if the time of occurrence t_a of activity *a* falls into the time interval t_e - *tol*. In the former case, the system notifies the patient of the medicine intake if the desired activity is detected as starting, and in the latter case, the system dispenses the medication and reminds the patient to take medication at the completion of the activity *a*. If the medication intake event *e* does not occur within the tolerance period, the patient would have violated the prescription instruction which would trigger notifications to the patient as well as the doctor. Figure 4.11 shows a visual representation of these intervals.

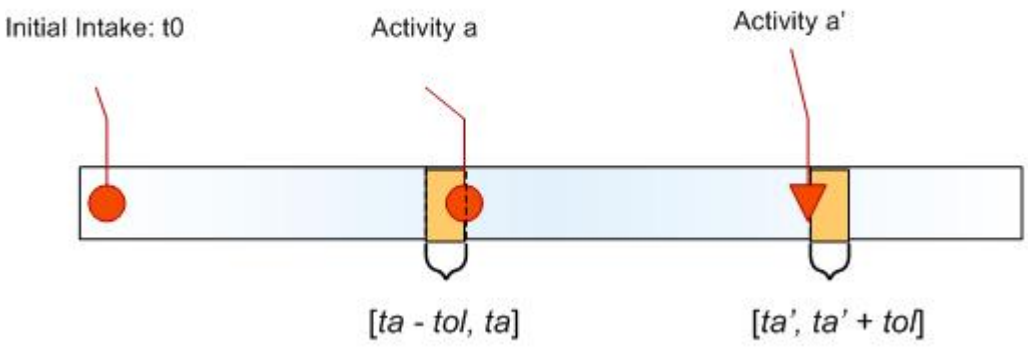

Figure 19 - Timeliness intervals: take m before activity a, take m' after activity a'

4.9 Prototyped Implementation

The prototype implementation in our SH Lab shows the feasibility of MISS. The Doctor, Pharmacy and SH Subsystems have been implemented as follows. An application for entering the prescription details, checks for conflicts, and store patient's record acts as the doctor's module. This node forwards the data to another node which corresponds to the pharmacy. The pharmacy receives the data, prepares the prescription and assigns an RFID tag to the prescription. Another node acting as the SH, queries the pharmacy's using the

RFID tag as the key and downloads the specific information about the prescription. Phidget RFID reader is used to perform the reading [72]. Different RFIDs assigned to several containers were tested as a patient might be taking different medications. This data the SH stores it in its database to use it to update other subsystems such as the reminder, notification and medication inventory. When a conflict is detected the corresponding message is displayed and notifications are given [68]. The Smart Home Subsystem was developed as a bundle that runs is OSGi, a framework particularly suitable for SH applications [46]. The flow of data was correctly transferred from the Doctor's node all the way to the SH and the experimental conflicts were correctly detected.

To have interoperability uses WS to communicate among subsystems. The DS uses WS in to access the third-party conflicts definition database. It also uses WS to communicate and send the data to the pharmacy. A WS is also provided so that a patient can have access to his or her record at any given time with the appropriate authentication mechanism. Likewise the pharmacy makes a similar use of the WS technology. It accesses the third-party conflicts definition database via WS. It also provides a WS for the doctors to establish a connection and forward prescription data. It also provides another WS so that the patient can access his or her record at any given time with the appropriate mechanism.

The SS makes use of the WS provided by the DS and the PS in order to keep a more complete record of the resident's medications, health conditions so that a final and more thorough check for conflicts can be performed at the house. It is made this way to respect the privacy concerns and laws such as HIPAA in case data is forwarded to the wrong party. The SH also has a mechanism to check not just for conflicts but also with compliance with the medication instructions. Timeliness and compliance requirements are enforced by using the prescription's data to define what to monitor, and the context information provided by the SH. Also because the pervasiveness of RFID remains in the future, we developed two parallel implementations: one using barcodes another one using RFID. This allows MISS to be used with current barcode technology and with the future RFID technology.

4.10 Conclusion and Future Work

The SH is equipped with technology to help the elderly and person with special needs perform activities of daily living, live more independently, and stay home longer. One vitally important activity of daily living is the management of medications. Several solutions have been proposed to help with medication management, but they tend to be non-networked and neglect the safety problems that arise from taking multiple medications, such as having medications interact with each other and knowing when to take each medication. This chapter presents MISS, a more comprehensive solution for successful medicine management that uses web services to provide interoperability among independent and existing medical systems and can detect conflicts among a patient's medications, health conditions and foods. Because patient's preferred doctors and preferred pharmacies may vary, the safety checking is enforced at multiple levels to catch more types of conflicts. We also investigate compliance with several types of prescription dosing instructions by monitoring intake actions with respect to the timeliness and completeness of each dose. We use modeling techniques for both the safety checking and for the compliance monitoring.

Our future work for this system includes completing the integration between OSGi and WS as well as the implementation of the compliance checking routine. We are also applying formal techniques to further model and verify the compliance and safety of the entire system and are looking to perform a case study with medical experts to validate the usefulness of our design.

CHAPTER 5. A COMPOSITION FRAMEWORK FOR SERVICES OF HETEROGENEOUS SERVICE-ORIENTED ARCHITECTURES

5.1 Abstract:

Service-Oriented Architecture (SOC) is a paradigm widely used to ease the development of applications such as those in pervasive spaces. Within the SOC paradigm, different Service Oriented Architectures (SOAs) have been developed. It is often the case that required services are implemented using different SOAs. Most composition frameworks take a two-tier approach: those that support a single SOA take advantage of service compositions and runtime substitutions, while those supporting services of different SOAs usually manage their interactions with manual service invocations. Some SOAs offer a set of features that others do not. In this chapter, we present a platform, language, and SOA-independent composition framework. Our framework accepts candidate services of heterogeneous SOAs and provides the functionalities to support composition of these. A case study and a performance analysis are presented to evaluate our framework.

5.2 Introduction

service is defined as an autonomous, loosely coupled, platform independent entity that can be described, published, discovered, and invoked. Services can perform a simple computation or compose a set of services to perform a complex task [1]. Service-Oriented Computing (SOC) is a programming paradigm where functionality is provided as services and applications are created by combining these services in a certain logical way. Instead of depending on a single component as the case of stand-alone applications, several services are orchestrated to provide the desired functionality. A fundamental characteristic of SOC is the separation of the service implementation from the service specification. There exist several Service-Oriented Architectures (SOAs) for this purpose, which define the specification of services and the protocols to interact with them. SOAs are designed to be loosely coupled, platform-independent and language-independent. Theoretically, services should be able to interact with other services as long as they have compatible interfaces. However, in practice it is difficult for services implemented using different SOAs to interact as each SOA definitions

are independent and their services depend on those specific definitions. Interactions and composition of services are possible within the same SOA but are difficult across different architectures.

Most research in SOA has focused on investigating web services (WS) [2]. Consequently, WS technologies and protocols are well studied and defined. Nevertheless, there exist several other SOAs such as Jini, Open Services Gateway initiative (OSGi), and Universal Plug-and-Play (UPnP). Certain properties of these other SOAs make them a better choice for certain applications, such as Jini's modular co-operating services for construction of distributed systems, OSGi bundles to control embedded devices, and UPnP services for plug-and-play networking with devices. To realize the full potential of the SOC paradigm, it should be possible to create composite services without depending on a particular SOA. We believe that the SOA standard should be able to accommodate and utilize different services, regardless of the particular languages, platforms, and architectures they are based on. Doing so could have a positive impact in the quality of service-oriented applications. For example, it can increase the number of potential services, which might impact the availability of services. Having more services available can also increase the number and the diversity of potential applications. Having a greater number of services to perform certain actions can improve the quality of the composite service by choosing those that better meet certain quality criteria. Additionally, composition of heterogeneous SOAs services can influence the performance of the application, as the services chosen can be those that show better performance without limiting only to those of a particular SOA. It can also improve reliability and substitutability, by having a set of services that provide the same functionality but implemented using different SOAs. In the case of service failure another equivalent service can be selected and still be able to perform the required task.

The contributions of our work include outlining and justifying a set of minimum requirements for compositions. These requirements are elicited based on a study of the standardization and composition strategies for SOAs. We believe that there is a great need of specific requirements especially in the area of compositions of services from heterogeneous SOAs and for the scripting languages used to specify composites. We contribute and define

the Simple Service Composition Language (SSCL), a language compliant with these minimum set of requirements and inspired in technology currently under use. SSCL is an XML-based language used to specify the workflow of composite services in a simple way that provides SOA independence, a key attribute for heterogeneous SOAs compositions. Our major contribution is an automatic composition framework that accepts a composite service description in SSCL as input and automatically produces an implementation of the composite service. The framework automatically searches the services needed, performs a safety check, creates the composite service implementation relying on services of heterogeneous SOAs, deploys the composite service, and starts it. One crucial step for the automation of our framework is having a service registry with all the candidate services. A customized registry based on the Universal Description Discovery and Integration (UDDI) for WS is provided that allows storing the information of services of different architectures. We also contribute with a context-aware mechanism for our framework that allows it to select among the services already in the context or search for services in the repository, and choosing the candidate service that improves performance when several services match the search criteria. The performance of the services is determined by studying the different SOAs, performing different experiments and studying the obtained results. An implementation of our composition framework is provided that supports OSGi and WS as an example. A guideline on how to extend the composition framework to support other SOAs is provided.

The feasibility of our composition framework is shown in a case study that compares the automatically generated composite service with current SOA composition strategies. The example used is based on the Medicine Information Support System (MISS) for Smart Homes [3]. As pervasive environments like the Smart Homes relies on the use of services, we believe this is an appropriate scenario. Several experiments are designed and conducted to measure the performance of the composition framework and a summary of the results is presented.

5.3 Related Work

Composition of WS is a topic that has been studied in the last few years yielding a series of strategies and tools to make it possible, yet several challenges still need attention [4]. Some

of these challenges are coordination, transaction, context awareness, conversation handling, execution monitoring, and infrastructure. Service compositions can be achieved using different strategies such as static, dynamic, model driven, declarative, automated, manual, and context based. In the literature, different scenarios are found that motivates the interest in performing service compositions as well as several frameworks that manually, semi-automatically, or automatically create composite services. Examples of these composition frameworks are e-flow, MAIS, MOEM, SELF-SERV, OntoMat-Service, SHOP2, WebTransact, and StarWSCoP [5]. Dustdar and Schreiner point out that no single composition framework satisfies all the needs for automatic WS composition [4]. The reason is that some frameworks implement a set of features or focuses on a particular issue overlooking the others. A more inclusive composition framework is still needed. In our work, we propose an automatic composition framework that tackles several of these issues by using a combination of strategies. Our framework pays special attention to infrastructure and context-awareness using dynamic, declarative, and context-based strategies.

As the number of individual services grows and applications become more complex, automated composition of services becomes more critical. In many cases, an individual WS does not completely satisfies certain requirements but when combined with other WS in a logical order, the composite service does. Currently, finding these individual services is a challenge as more organizations provide functionalities as new WS, resulting in an explosion of the number of services available. Searching for the appropriate services and ways to combine them has become a problem beyond the human capability due to the size of the search space and the dynamicity of services as they are added, removed, or updated on the fly. The need for automatic WS composition becomes evident and critical. Rao and Su in [6] survey two main approaches to achieve automated WS composition: workflows and artificial intelligence (AI) planning. With the workflow approach, the user provides a workflow of the intended composite service and the system automatically locates the individual services that satisfy the workflow execution. Some platforms that use the workflow approach are e-Flow [5] and Polymorphic Process Model (PPM) [8]. With the AI planning approach, the developer gives a set of constraints and preferences and the system generates the flow and

finds the candidate services [9]. The strategies used in AI planning include situation calculus [10], planning domain definition languages (PDDL) [11], rule-based planning [12], and theorem proving [13]. Some of the tools available that support AI planning service composition are SWORD [14], SHOP2 [15], and DAML-S (also known as OWL-S) [16]. No matter what composition approach is chosen, they all should provide the presentation of a single service, translation of the languages, generation of composition process model, evaluation of composite service, and execution of composite services. One observation is that most of these composition frameworks use abstract representations of services. In our work, a framework is provided that composes actual services using workflow techniques.

Several research works have proposed solutions for the problem of automatically composing services in other SOA such as OSGi. Wood et. al. in [17] presents 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. Redondo et. al. in [18] use the Business Processing Execution Language for Web Services (WS-BPEL), to describe the workflow of OSGi composite services. They provide a framework that extends OSGi by allowing it to interpret WS-BPEL files, locate the services, and create the composite service. Their framework adds tags to the WS-BPEL file and focuses on composing OSGi services only. Anke and Sell in [19] present a strategy for automatically composing OSGi services by first converting them to WS and then use WS composition techniques. Our work is different, as we want to compose services of heterogeneous architectures in their original form without the need for converting them.

Other researchers have focused on the problem of composing heterogeneous SOAs. Lee et. al. in [20] support WS-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 WS-BPEL file. These tags specify the service type (OSGi or WS) and the binding information like the URL of the WSDL file or the location of the OSGi bundle. Our approach is different as the service type and binding information are automatically gathered from a service repository without the need for extra tags. Therefore, the service specification file needed in our framework is simpler, less verbose, and more interoperable. 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) [20] found in frameworks such as Apache Tuscany [21], Fabric3 [22] and the Newton Framework [23]. 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 in which the developers can create composites using services with heterogeneous implementations. They rely on annotation and a special XML language, the Service Component Definition Language (SCDL), to define the components and the composites. The problem with SCA is that the tags and annotations have to be included in the source code of the services for the framework to accept them. This limits the services that can be used to only those developed under a SCA infrastructure. Our work is different as we use the services in their original implementation, allowing to reuse services already available and without the need to add annotations to the source code.

5.4 Requirements for SOA Compositions

The composition framework is required to be platform, language, and SOA independent. It should support heterogeneous SOAs and automatically create composite services. Even when the SOC standard already provides for platform and language independence, composition strategies are usually limited to services belonging to the same SOA. Other composition strategies that support multiple SOAs such as the SCA [24] use annotations to connect the services together. This approach limits the set of services that can be used to only those with the appropriate annotations. A composition framework should respect the fundamental SOA requirements of separation of service specification from the implementation. A composition framework should offer language independence, SOA independence and not depend on the use of annotations in the source code of the service.

Several standardization bodies make efforts to come up with a SOA standard but there is a need for refinement on some aspects of this standardization. The OASIS-OPEN has defined the Reference Model for Service Oriented Architecture [25] to encourage the continued growth of different and specialized SOA implementations whilst preserving a common layer of understanding about what SOA is. In their Reference Model, they define the techniques

for composition of services, with orchestration as the main methodology. They define orchestration as “a technique used to compose hierarchical and self-contained service-oriented business processes that are executed and coordinated by a single agent acting in a ‘conductor’ role”. These orchestrations are automated by using a business process orchestration engine. These orchestration engines are hardware or software components that act as the central coordinators for executing the flows that comprises the orchestration. The flow for the orchestrations typically is described using a scripting language. The SOA standard assumes that the software or hardware orchestration engine executes the process flow specified in the scripting language. However, this standard only gives a high-level definition and does not provide specific requirements for the scripting language or for the orchestration engine. Therefore, we studied the SOA standard and several SOAs to identify this set of minimum requirements.

We carefully study the characteristics of various SOAs, focusing especially on WS and OSGi architectures and looking for desirable features for our composition framework. WS has been the main subject of research in the SOA community to the point that some works use the terms SOA, SOC, and WS interchangeably. Consequently, WS is a strong, widely adopted technology whose protocols, tools, and strategies are well defined. One of these WS protocols is WS-BPEL, a language for describing composite services and workflows. This protocol has become the de facto standard for WS compositions. Several orchestrations engines are examined that use WS-BPEL as their input language. Other important SOA is OSGi, whose standard defines a framework for the deployment of services packaged in deployment units called bundles. Several open-source and commercial implementations of the OSGi framework are available such as Eclipse Equinox [26], Knoflerfish [27], and Apache Felix [28]. One of the main features of the OSGi standard is the security layer it provides, as the OSGi framework runs as an isolated Java Virtual Machine. The OSGi framework provides dynamic installation and removal of bundles without the need to stop or restart the system. The OSGi standard defines a set of default bundles including a HTTP server and log service that are useful to support other SOAs.

A set of essential features have been identified for our heterogeneous SOA composition framework after studying the SOA standard and different architectures. Several essential features are shared by the different SOAs and orchestration methodologies, while some features are lacking in one SOA or the other. One of these essential features needed is a SOA-independent orchestration language for compositions. This composition language must support variables and standard data types such as string, integers, and Booleans. This composition language should also provide the means to specify the candidate services using their interface name. The language should also support sequential execution of commands including invocation of a service, variable assignments, decision structures and a repetition structures.

An orchestration engine that understands this language and creates the implementation of the composite service is needed. The orchestration engine should be capable of supporting services from heterogeneous SOAs and to create the composite service implementation transparently and automatically without requiring the user intervention. The performance of the composition framework is very important. Therefore, the overhead added should remain low and the framework design and implementation should maximize services performance. The services performance is determined based on the features of each SOA and by experimental results. The next section describes the syntax of our proposed new language used for describing composite services.

5.5 The Simple Service Composition Language (SSCL)

In this section, we define the syntax of the Simple Service Composition Language (SSCL). The language SSCL is inspired on the syntax of the Business Processing Execution Language for WS (WS-BPEL) and the simplicity of the Service Component Definition Language (SCDL) for SCA. This new language is different especially as it provides a simplified and SOA-independent way to specify composite services workflows. SSCL is straightforward to learn and has the necessary set of commands to perform the functionalities required for composition of services. Developers that already know WS-BPEL and SCDL will find SSCL familiar while new developers should be able to learn it quickly.

The SSCL syntax follows their counterparts WS-BPEL and SCDL with several unique features that simplifies it and allows it to support heterogeneous services descriptions. One of these differences is the syntactical simplicity of the SSCL commands. The names of the commands and tags are similar to those found in WS-BPEL, but less attributes and tags are required. This makes SSCL less verbose and easier to learn and understand. SSCL supports commands and tags that are familiar to developers of many programming languages such as *if, variables* and *while*, but does not support SOA specific commands and tags such as WS-BPEL *reply, flow*, and *correlation* or SCDL *implementation, binding,* and *reference.* This makes SSCL more independent, simpler, and convenient. With SSCL, variables can be specified as a comma-separated list of variables in a single attribute entry. In WS-BPEL one attribute has to be specified for every variable, which makes the specification file more verbose, confusing, and harder to parse and process. Another difference is that SSCL supports Java-style specifications for logical symbols. The comparisons can be specified using the familiar symbols <, <=, ==, >, >= and != just like in many other programming languages, while in WS-BPEL these are specified using special codes. This makes SSCL easier to write and read. The skeleton of a SSCL file is provided next, followed by the details for each command:

SKELETON OF A SSCL FILE:

```
<process name = “process_Name” tagetNamespace = “targetNamespace_URL” >
    <partnerLinkTypes>
        <partnerLinkType name = “service_ interface_ name”/> …
    <partnerLinkTypes>
    <partnerLinks>
    <partnerLink name="service_name" partnerLinkType=" service_ interface_ name"/> …
    </partnerLinks>
    <variables>
        <variable name="variable_name" type="xsd:TYPE"/>…
    </variables>
    <sequence name = “main”>
```

```
A SET OF THESE COMMANDS
<invoke partnerLink="service_name" operation="method_name"
inputVariable="input_variable(s)"  outputVariable="output_ variable(s)" />
<assign>
        <copy>
                <from>variable_name or value</from>
                <to variable="variable_name"/>
        </copy>
</assign>
<while >
        <condition>   CONDITION  </condition>
        ACTIONS…
 </while>
<if >
        <condition>   CONDITION </condition>
        ACTIONS…
<elseif>
        <condition>   CONDITION </condition>
        ACTIONS…
</elseif>…
<else>
        ACTIONS…
</else>
</if>
<repeatUntil>
        ACTIONS…
        <condition>   CONDITION  </condition>
</repeatUntil>
<forEach counterName="N">
        <startCounterValue> VALUE </startCounterValue>
```

```
            <finalCounterValue> VALUE </finalCounterValue>
            <scope> ACTIONS … </scope>
        </forEach>
    </sequence>
</process>
```

The first tag in SSCL is the *process* tag that has two attributes: *name* and *targetNamespace*. The mandatory *name* attribute indicates the name of the composite service that the composition framework will later use to define the composite service interface. The optional *targetNamespace* attribute indicates the URL of a namespace with qualifying elements and attribute names.

The *partnerLinkTypes* tag encloses a series of *partnerLinkType* sub tags. Each *partnerLinkType* sub tag has a mandatory *name* attribute that indicates the name of a different service interface. The composition framework uses these service interfaces to select the candidate services.

The *partnerLinks* tag encloses a series of *partnerLink* sub tags. Each *partnerLink* sub tag defines an instance of a service. The *partnerLink* has two mandatory attributes: *name* and *partnerLinkType*. The *name* attribute indicates the name of the instance of the service and the *partnerLinkType* indicates the service type.

In SSCL, variables are defined with the *variables* tag that encloses a series of *variable* sub tags. Each *variable* sub tag defines a unique variable and has two mandatory attributes: *name* and *type*. The *name* attribute indicate the identifier for the variable and must be unique. The *type* attribute indicates the data type of the variable. SSCL supports standard data types such as integer, Boolean and string, and others that are part of the XML Standard Data Types [106].

The orchestration logic that supports sequential execution of commands starts with a *sequence* tag, which is mandatory and has a *name* attribute. Usually the value "*main*" is

assigned to the *name* attribute, indicating the beginning of the orchestration logic. The *sequence* tag encloses a series of commands that define the composite service operation.

One of these commands supported is *invoke,* used to make calls to the methods of a services. It has four attributes: *partnerLink, operation, inputVariable, and outputVariable*. The *partnerLink* attribute indicates the name of the service to invoke, as defined in the *partnerLinks* tag. The *operation* attribute indicates the actual method of the service to invoke. The *inputVariables* attribute indicates the variables that the method takes as arguments. The *outputVariable* attribute stores the result returned by the service after the method invocation.

The *assign* command is used to copy the content of a variable or a constant value into another variable. *Assign* does not have any attributes but it has a *copy* sub tag, which includes two sub tags: *from* and *to.* The *from* sub tag indicates the variable or constant to copy the value from, while the *to* sub tag indicates the variable where to copy to. The *to* sub tag has the *variable* attribute that indicates the name of the variable to copy to as defined in the *variables* tag.

The *if* command is a decision structure that executes a series of actions depending whether a Boolean condition evaluates to true or false. The *if* tag have several sub tags. The *condition* sub tag indicates the condition criteria, which can be a Boolean variable or a Boolean expression. Following the *condition*, the *ACTIONS* block indicates a series of commands to execute when the condition evaluates to true. There is also the *elseif* sub tag that evaluates another *condition* in case that the previous one evaluates to false, with its corresponding *ACTIONS* block. Several *elseif* sub tags may be included. Finally, the *else* sub tag *ACTIONS* block is executed if all previous *conditions* evaluate to false.

The *while* command is a loop structure that executes a series of statements while a condition remains true. The *while* tag has a *condition* sub tag and an *ACTIONS* block. The *condition* sub tag indicates the condition criteria to evaluate before entering a new iteration. The *ACTIONS* block contains the series of commands to execute while the condition remains true.

The *forEach* command is a loop structure that executes a series of commands a predetermined number of times. It has the attribute *counterName,* and the sub tags *startCounterValue, finalCounterValue,* and *scope*. The *counterName* attribute indicates the variable that will keeps track of the iterations. The *startCounterValue* attribute encloses the initial value for *counterName*. The *finalCounterValue* attribute encloses the final value for *counterName*. The *scope* sub tag encloses an *ACTIONS* block with the set of commands to execute.

The *repeatUntil* command is a loop structure that executes a series of commands at least once and until a condition evaluates to true. It has an *ACTIONS* block followed by a *condition* sub tag. The *ACTIONS* block indicates the set of commands to execute. The *condition* sub tag indicates the condition to evaluate.

As already mentioned, standardization effort has focused mainly on WS standards with other efforts in technologies like SCA and OSGi. However, all of them have failed to define a general guideline for compositions of services from heterogeneous SOAs. We believe SSCL will be of great help to filling this gap, and SSCL can be part of the SOA standard as it provides the functionality needed, it is easy to understand, and provides SOA- independence. The formal syntax of SSCL is presented in Appendix A.

5.6 The Composition Framework Details

At a high level, our composition framework works as follows: accepts a composite service workflow description in SSCL, locates the candidate services from the context information or by looking into the service repository, models the service specification, checks safety criteria, automatically creates the composite service implementation, and installs the new composite service. The composition framework is developed under the OSGi platform implemented as an OSGi bundle, providing support for OSGi services composition. To provide support for WS composition, a bundle that handles all WS protocols is developed. By combining them our framework provides support for heterogeneous SOAs, using OSGi services and WS as examples. The composition framework uses several other bundles and resources in order to support SSCL, manage the service repository, find services already in use, model check the

service specification, and create the actual implementation. Our composition framework combines several WS and OSGi protocols and adjust them to support both technologies and allow composing heterogeneous SOA services. The result is a composition framework implemented as an OSGi bundle that supports seamless composition of SOAs. Our composition framework automatically performs all the underlying processing, maintains the OSGi services and the WS in their original implementation and produces the composite service implementations as OSGi bundles. This differs from previous solutions that transform services into another SOA or add tags to the services source code. Figure 5.1 depicts the architecture of our composition framework**.** As it accepts a SSCL file and produces an OSGi bundle, the framework is named SSCL2OSGi. The next sections summarize the other bundles developed, the modifications needed, and a guideline to expand the framework to accept other SOAs.

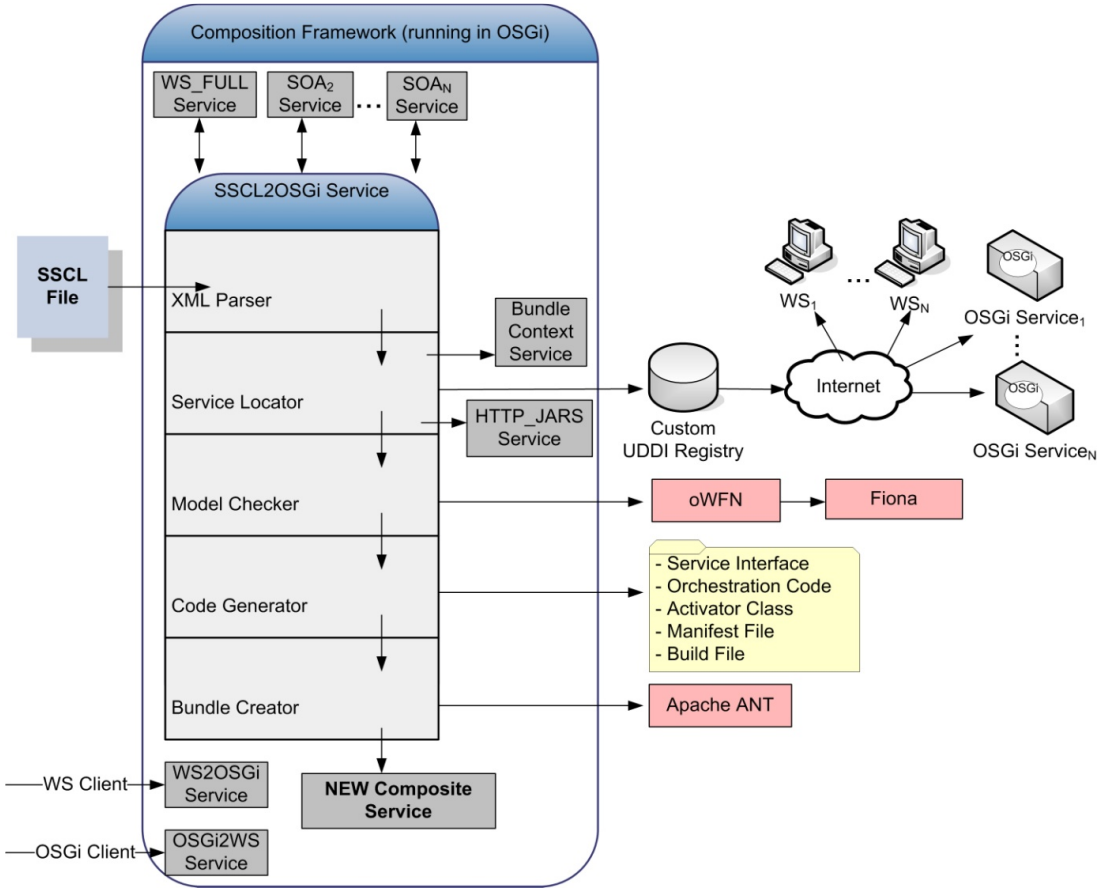

Figure 20 - The SSCL2OSGi Composition Framework Architecture.

5.6.1 WS_FULL Service:

Our framework focuses on creating composite services based on the functionality desired and not on a particular architecture. The interactions with WS are examined in order to provide full support to this SOA. Our main interest is to have OSGi services to be able to invoke WS and vice versa. In our early experiments, we developed WS composites as stand-alone applications and wrapped them as OSGi bundles. As stand-alone applications they run as expected, but the wrapped OSGi version once deployed into the OSGi framework does not run properly. Intuitively, invoking WS from the OSGi should be a straightforward process as the OSGi framework is a Java Virtual Machine, the OSGi bundles are Java JAR files, and Java tools are used to generate the WS stub code. However, a “bundlefied” Java-based WS has run-time issues with the OSGi framework.

To interact with the WS, the Eclipse IDE Web Service Client wizard is used to generate stub code that abstracts the SOAP messages communication. The WS stub code is wrapped as an OSGi bundle [107] and deployed into the OSGi framework [40]. The knopflerfish Axis bundle is used to provide WS support. However, deploying the wrapped WS resulted in a series of errors. We studied the knopflerfish Axis bundle and found that it uses the Apache Axis version 1.1 as its SOAP engine, while most recent WS implementations use Apache Axis version 1.4. There are significant differences between Apache Axis 1.1 and version 1.4 such as the naming convention, package names and keywords, in part because Axis 1.1 uses Java 1.4 naming convention while Axis 1.4 uses Java 1.5/1.6 [108]. After figuring out these problems, it was possible to invoke some WS methods but not others.

We find out that there are different document styles used by the different versions of SOAP [109]. Apache Axis 1.1 only supports Remote Procedure Calls (RPC) and RPC/encoded document styles, while Apache Axis version 1.4 supports RPC, RPC/encoded, document/literal and document/literal encoded styles. Support for these different styles is provided to ensure that the composition framework calls WS using the correct one [110]. An appropriate version of the knopflerfish OSGi framework that supports our version of the WS

technologies was found. Several OSGi-wrapped WS using the different document styles were tested successfully and support for the WSDL protocol also included.

Support to the UDDI protocol was added in order to have the complete WS protocol stack. This support is needed to query services repositories to find services description and binding information. For supporting this protocol, the UDDI4J API [37] was wrapped into our service with a simplified interface. After all these steps, the WS_FULL bundle was created to provide support to the full WS protocol stack: SOAP, WSDL, and UDDI. To the best of our knowledge, this is the first comprehensive solution that supports the entire WS protocol stack within OSGi. This bundle also provides a SOAP engine that allows OSGi services to be exported as WS.

5.6.2 SSCL2OSGi Service:

The actual composition framework is implemented using OSGi technology. The composition framework is named SSCL2OSGi as it accepts as input a composite service workflow specification file in SSCL and automatically produces an OSGi service that implements it. The overall steps of SSCL2OSGi are the following:

- Step 1: *Parses the SSCL file*
- Step 2: *Gets candidate services already in the system context*
- Step 3: *Gets the remaining candidate services from the service registry*
- Step 4: *Translates the SSCL orchestration logic into executable Java code*
- Step 5: *Creates the composite service implementation as an OSGi service*
- Step 6: *Installs the service into the OSGi framework and starts it*

Now each of these steps is described in detail:

Step 1: *Parses the SSCL file.* The SSCL2OSGi service takes as input the workflow specification of a composite service in SSCL. During this step SSCL2OSGi checks the XML-syntax, parses the SSCL file, analyzes its content and creates the parse tree.

Step 2: *Gets candidate services already in the system context.* From the parse tree, SSCL2OSGi navigates the *partnerLinkTypes* tag. This tag groups a set of *partnerLinkType* sub tags, each of them containing the interface name of an individual service. Using the service interface name, SSCL2OSGi proactively searches for those services already in the context using the OSGi BundleContext interface. The composition framework selection criteria choose first these services.

Step 3: *Gets the remaining services from the service registry.* Those services not found in the system context are searched in the service registry. Using the service interface name, this registry is queried to find service details such as its description, URL, and service type. As the service registry should support different SOAs, a customized version of UDDI was used. We customized it by carefully studying UDDI and noticing that it provides a data structure called *tModels*, which are a generic mechanism for describing technical details about WS. We used these *tModels* to describe and register the services, and defined several *tModels* to indicate services details such as their type and remote location. By relying on *tModels*, the customized registry is able to register details of heterogeneous SOAs such as WS and OSGi and it can be extended to support others by adding the appropriate *tModels* entries.

Step 4: *Translates the orchestration logic of SSCL into executable Java code.* In SSCL, the *partnerLink* tags describe instantiations of the service indicated in the *partnerLinkType* attribute. These are translated into Java by taking the *name* attribute as the object identifier and the *partnerLinkType* attribute as the class it instantiates.

Each *variable* tag define a unique variable and is translated into Java by taking the *name* attribute as the identifier and the *type* attribute as the corresponding Java data type.

The *sequence* tag indicates the initial method that in Java corresponds to the *main* method.

The *invoke* tag translates into a Java method call by taking the *partnerLink* attribute as the object, the *operation* attribute as the method to be invoked, the *inputVariable* attribute as the variable or variables expected as input, and the *outputVariable* attribute as the variable where to store the result returned.

The *assign* operation translates into a variable assignment statement in Java by taking the *from* sub tag as the variable or value to be assigned, and the *name* attribute of the *to* sub tag to indicates the variable where to assign the value.

The *if* operation translates into an if decision statement in Java by taking the *condition* sub tag as the Boolean expression to be evaluated, the *ACTIONS* block as the commands in the body of the if, the *elseif* and *else* tags as their corresponding Java expressions.

The *while* operation translates into a while loop statement by taking the *condition* sub tag as the expression to be evaluated and the *ACTIONS* block as the commands in the body of the loop.

The *forEach* operation translates into a for loop statement by taking the *counterName* attribute as the counter variable, the *startCounterValue* as the initial value for *counterName*, the *finalCounterValue* as its final value. Finally, the *ACTIONS* block within the *scope* sub tag is translated as the commands in the body of the loop.

The *repeatUntil* statement translates into a do while loop by taking the *ACTIONS* block as the body of the loop and the *condition* tag as the Boolean expression to be evaluated.

Step 5: *Creates the composite service implementation as an OSGi bundle.* After translating the orchestration logic into executable Java code, the composition framework creates the service interface, the manifest file, the activator class, and the ANT build file required for OSGi bundles using the information extracted from the SSCL file.

Step 6: *Installs the bundle into the OSGi framework and starts it.* After creating the executable Java code and the other required files, Apache ANT is used to compile the files and check the dependencies. If all dependencies are resolved and no syntax errors are found,

the composition framework creates a bundle that packages all into a JAR file. The composition framework automatically deploys the JAR file into the OSGi framework, and starts it. After this point, the new composite service can be used.

5.6.3 HTTP_JARS Service:

The OSGi standard requires a simple HTTP service but our composition framework requires certain features that are not provided by it. To add the remaining features needed the HTTP_JARS service was created. With this service, the OSGi framework serves as a web repository where services can be listed, located, and downloaded. This is especially useful when registering OSGi services into the service registry as services now running in the framework can be accessed and downloaded using a relative path. The HTTP_JARS service combined with the WS_FULL and the OSGi default HTTP service provides all the support needed for WS operations.

5.6.4 The OSGi2WS Service:

As an optional service, the OSGi2WS is provided to allow WS developers to utilize OSGi services easily and quickly. The OSGi2WS service is a proxy factory that creates proxy services that export OSGi services as WS. The proxy created sits between the client and the original OSGi service and provides WS functionalities like WSDL description and SOAP messages exchange. Developers will be able to invoke OSGi services as if they were WS. The OSGi2WS service extracts the Export-Packages from the manifest file and imports these packages into the proxy service. It adds a reference to the original OSGi service into the Activator class to export it as a WS. Finally, it packages everything into an OSGi bundle and installs it automatically. The OSGi2WS service has a service listener that detects when new OSGi services are deployed and prompts the user whether to create a WS proxy. The OSGi2WS service can also be invoked directly by indicating the target OSGi service.

One of the benefits of the OSGi2WS service is that OSGi and WS developers can work together as services can be represented in both technologies. OSGi developers can focus on creating their services and export them as WS via a proxy. Another benefit is the separation of duties, as the proxy bundle is independent and can be stopped or removed while still

conserving the original OSGi service. Creating proxy WS is not without cost as the OSGi2WS requires an extra bundle and an additional layer of communication. This can potentially degrade performance when compared to direct invocations to the original OSGi service. Still the OSGi2WS provides a straightforward way to export OSGi services as WS. Our composition framework focuses on using services in their original implementation. However, this service is provided as part of our comprehensive approach and to support previous solutions that convert OSGi services into WS.

5.6.5 The WS2OSGi Service:

Another optional service with a similar motivation as the OSGi2WS service is the WS2OSGi service that creates an OSGi proxy of a WS. The WS2OSGi is also a proxy factory but it creates OSGi proxy services that communicate with the actual WS. The proxy created sits between the client and the WS and provides OSGi functionalities to invoke WS as an OSGi service. The OSGi2WS bundle takes as input a WSDL file, generates the stub code, creates the necessary OSGi files, packages the stub code and the generated files into an OSGi bundle, and automatically deploys it into the OSGi framework. The WS2OSGi service can also be invoked directly using as input the WSDL file of the target WS.

The WS2OSGi bundle offers the benefit of allowing WS and OSGi developers to work together making services available in both technologies. WS developers now have a mechanism to export their WS as OSGi services. The cost of this approach is perhaps slower performance when compared to direct invocations to the WS. Nevertheless, it gives developers a tool to simplify and diversify the services development process. The focus of our composition framework is to use services in their original implementation but this tool is also part of our inclusive approach, especially keeping in mind that most SOA developers use WS.

5.6.6 Guidelines for Supporting Other SOAs:

The SSCL2OSGi composition framework supports OSGi and WS as an example but can be extended to support other SOAs. For adding support of a new SOA, its functionality can be wrapped in any supported SOA, as is the case of the WS_FULL service described before that

was wrapped as an OSGi service. It is very important to resolve all packages dependencies, solve any versioning issues, include all the required libraries, and import/export all the necessary packages. One of the most important file that needs to be checked is the manifest file, especially the Import-Packages and Export-Packages entries. All the necessary libraries should be included in the classpath entry of the manifest. All these files should then be packed into a bundle jar file and installed into the OSGi framework. As the composition framework also relies on a customized UDDI registry, the corresponding *serviceType tModel* for specifying services in the new SOA should be created. New services in the new SOA must register in the UDDI using the new *serviceType* and provide the other service details using the appropriate *tModels*.

5.7 Case Study for MISS

The feasibility of our composition framework is demonstrated with a case study where the workflow of a service is provided in SSCL, the composite service is created, installed, and executed. The case study presented is based on the Medicine Information Support System (MISS) for Smart Homes [68] that helps patients with the management of their medications. MISS integrates the doctor, the pharmacy, and the smart home subsystems allowing information to be shared among them. Its main purpose is to increase safety by checking for conflicts between new prescription and previous medications, health conditions and food items. MISS performs the conflict checking using a trusted third party. By checking new prescriptions data with the patient's local record at each subsystem, MISS detects all possible conflicts.

The overall MISS functionality starts at the doctor's subsystem where the doctor enters the new prescription's data and checks for conflicts with previous medications and health conditions using a trusted third party [111]. If no conflict is found, MISS uses a secure communication channel and forwards the prescription data to the pharmacy. At the pharmacy subsystem, MISS checks for conflicts using the patient's local pharmacy record of medications and conditions. If no conflict is found, MISS uses a secure communication channel and forwards the prescription data to the smart home subsystem. At the smart home,

MISS performs a final check for conflicts this time using the patient's local smart home record of medications, foods, and conditions. Appendix B contains the SSCL file describing the workflow of MISS and figure 5.2 shows a diagram for this workflow. This figure shows the interactions between the pharmacy subsystem and the smart home subsystem. The process execution starts with the Receive Input operation. After that, the RFID Service is invoked to read the RFID tag of the medication. This RFID tag uniquely identifies the medication and it is used when the Pharmacy Service is invoked to query the prescription details. The Assign block assigns the values received from the Pharmacy Service into a set of local variables and pulls the local patient's record. The new prescription data and the local patient's record are used to invoke the MCD Service [111]. This service checks for conflicts among the new prescription with previous medications, health conditions, and food items at home. Depending whether there is a conflict, the Speech Service and Notification Service are invoked with the corresponding message. The Speech Service gives a spoken notification to the users in the home, while the Notification Service sends email, text messages or makes phone calls to the appropriate parties. Finally, if no conflicts are found the new prescription data is stored into the Smart Home subsystem when the Medicine SH Server service is invoked.

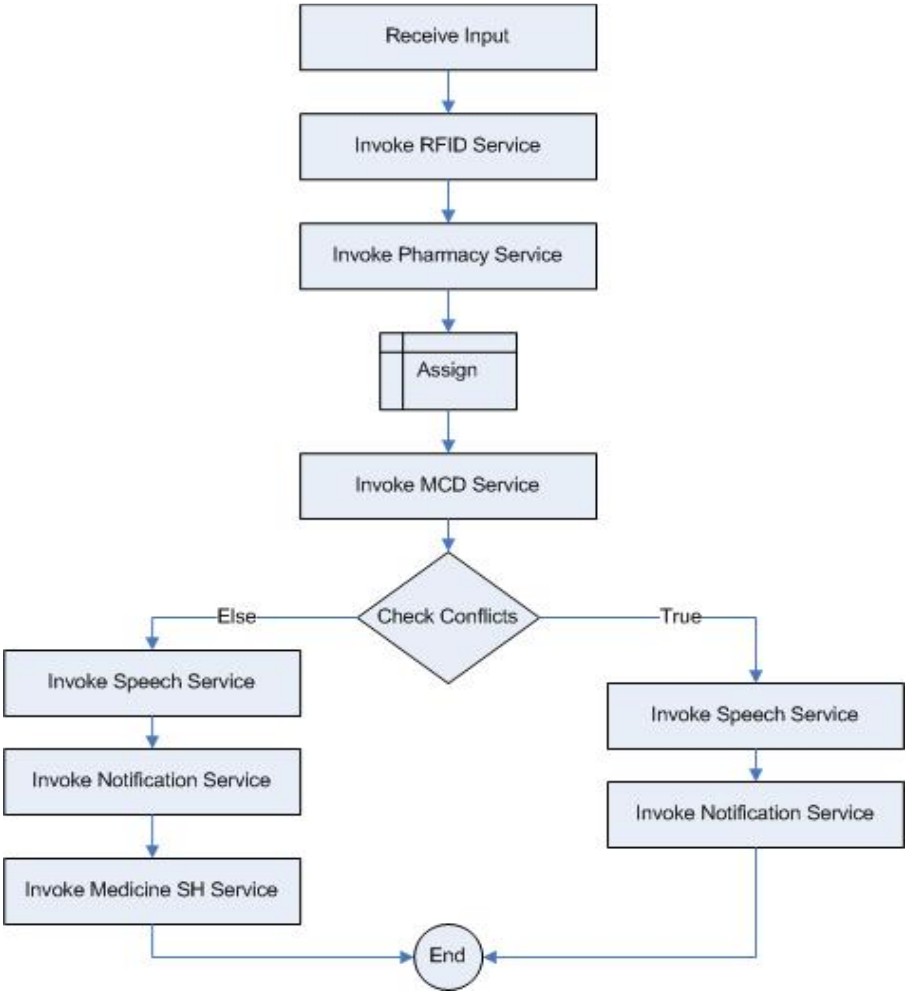

Figure 21 - Graphical representation of the workflow of the MISS Service using SSCL.

5.8 Evaluation

To determine the effectiveness of our composition framework the composition time overhead and the execution time of the created composite service are measured. A summary of the results is presented next.

5.8.1 Composition Time Overhead

We study the overhead added and the performance cost of the activities of our automatic composition framework. The major activities of the composition framework are Parsing, Model Check, CodeGen, and Create Bundle. Parsing measures the time it takes to read the

SSCL file and create a parse tree. Model Check measures the time it takes to check for syntax errors and standard logical errors. CodeGen measures the time it takes to translate the SSCL code into Java code and create the other necessary files. Create Bundle compiles, installs and starts the new composite service implementation. The time to create the composite service for the MISS workflow is measured. In the experiments, we create the composite service and measure the creation time at least 100 times. Figure 5.3 presents the average time taken by each major activity measured in miliseconds. It shows the overhead the composition framework adds for this particular example, and gives insight on how the framework might behave in general. This analysis helps to identify the activities that need improvement in order to reduce the execution time and improve performance.

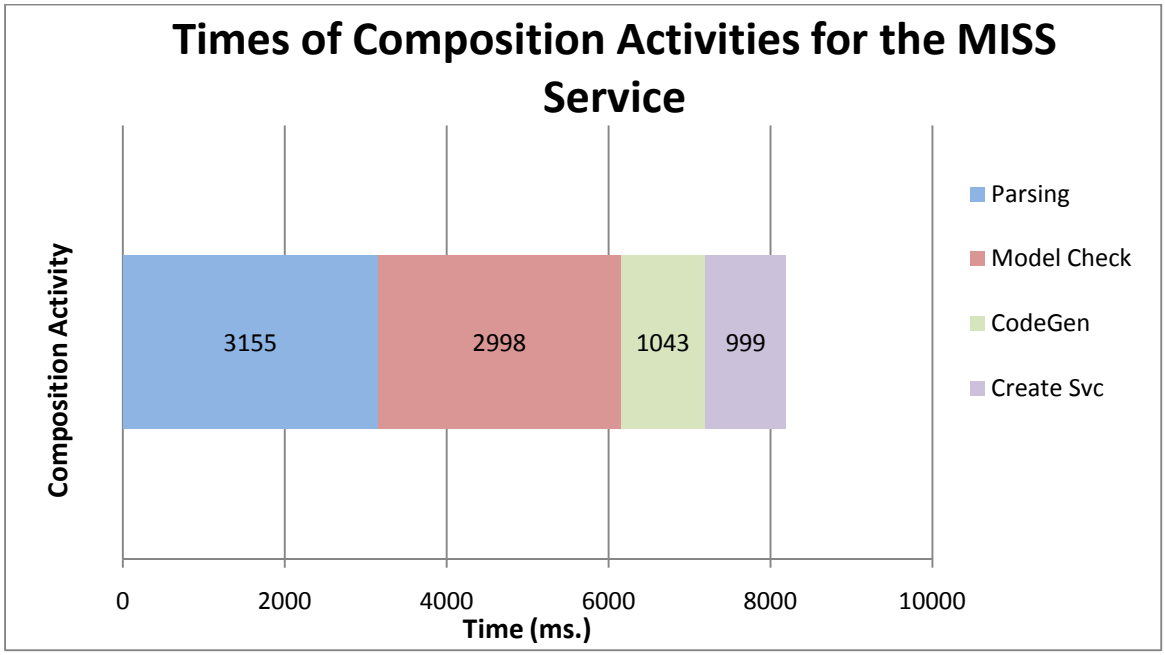

Figure 22 - Time of the composition activities for MISS.

As seen in Figure 5.3, the parsing process is the most expensive activity in terms of execution time. The parsing process is studied closer and the results are shown in Figure 5.4. The parsing process activities are Get Sequence, UDDI, Get PLT, File Download, WS Stub, and Service Impl. Get Sequence extracts the orchestration logic of a SSCL file. UDDI

measures the average time of a roundtrip query message to the UDDI registry. GetPLT measures the time to get the ParnerLinkTypes and bind to the actual services. File Download measures the average time to download the JAR file of an OSGi bundle. WS Stub measures the average time to get the WSDL file and generate the stub code when using WS. Service Impl measures the average time for generating the Java class that implements the orchestration logic. Figure 5.4 shows that generating stub code takes 500 milisecond, while downloading the jar takes much less. Response time of OSGi services is also much quicker than WS. Even when creating services using the composition framework adds some overhead, this is a one-time operation. There is a tradeoff between the advantages of using the automatic composition framework and the overhead added that must be considered. The next subsection presents an analysis on how the performance of the resulting composite service compares to the traditional composition methods for that use a single SOA.

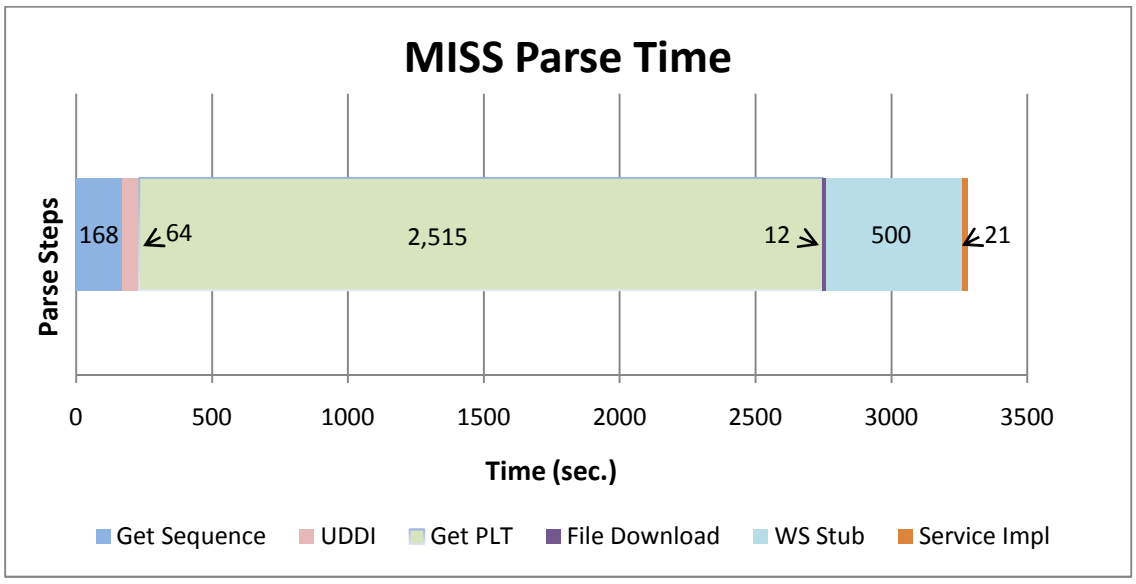

Figure 23 - Parse time of the SSCL file implementing the MISS Service.

5.8.2 Execution time

The execution times of composite services using current composition techniques are compared with the heterogeneous SOAs composite services automatically generated by our composition framework. In the experiments, we create a composite service that only uses OSGi services (MISS_OSGI), a composite service that only uses WS (MISS_WS), and the

third composite service that is automatically generated by the SSCL2OSGi composition framework that combines OSGi and WS (MISS_COMB). Our hypothesis is that MISS_OSGi will perform better, followed by MISS_COMB 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. Figure 5.5 shows the average time and the standard deviation results of the experiments after executing the three types of composite services at least 100 times, while Figure 5.6 shows the minimum and maximum values.

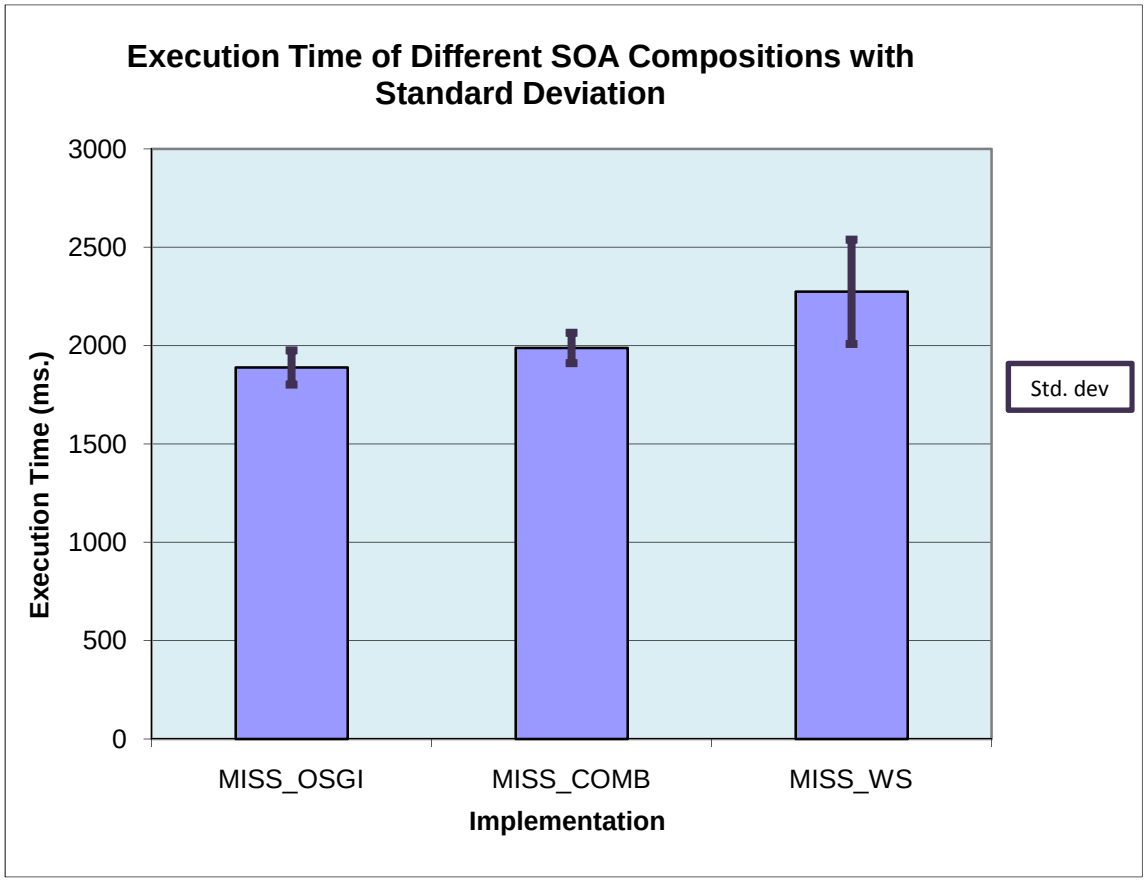

Figure 24 - Execution time of each implementation where the dark line indicates the standard deviation.

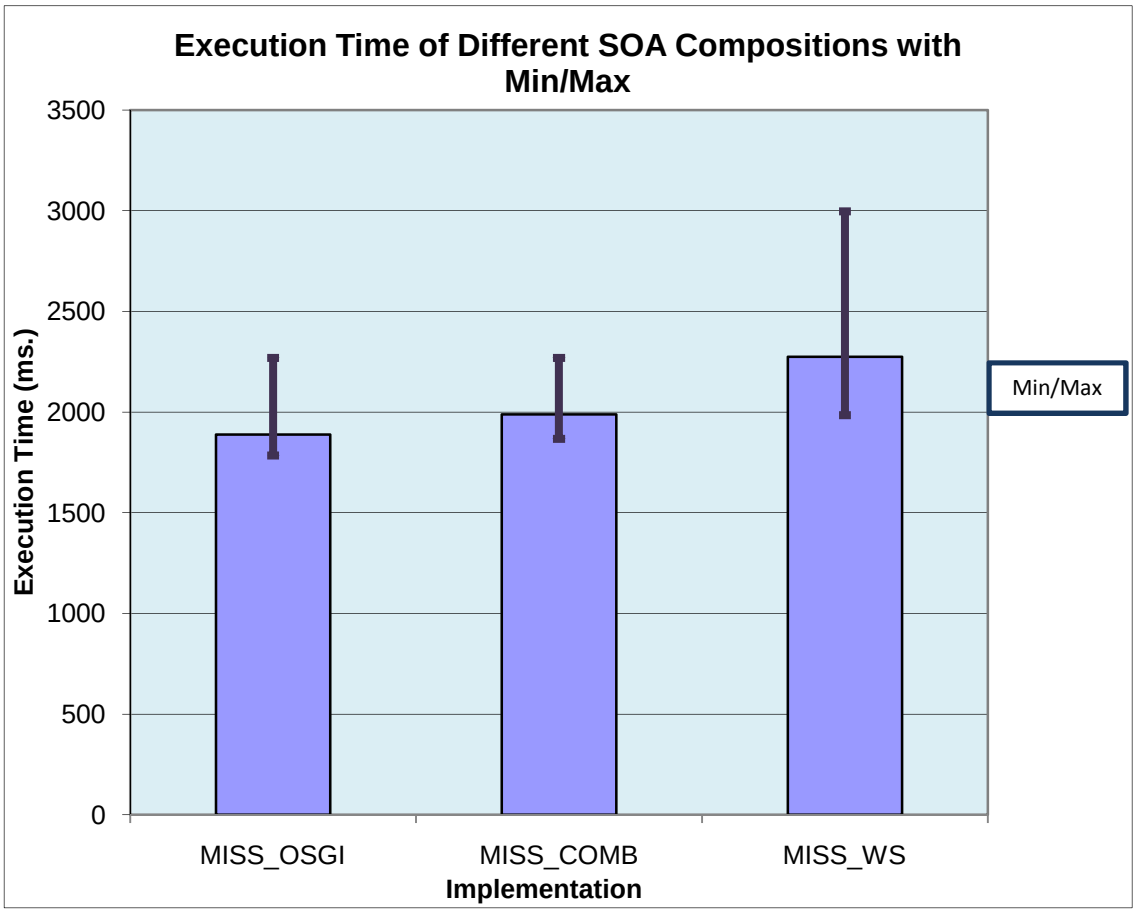

Figure 25 - Execution time of each implementation where the dark line indicates the minimum and maximum.

These results support our hypothesis as the MISS_OSGi has the best performance, followed by MISS_COMB and the composition with the worst execution time is MISS_WS. Nevertheless, the time difference among them is less than 400 milisecond. This amount of time for this particular application is acceptable and when observed by a human being it is almost unnoticeable. The fact that WS are the slower but the most popular and widely used SOA, gives us an upper bound on the time that is acceptable for these applications. Another positive result observed is a reduction in the execution time of the worst case when heterogeneous services are used. It is an advantage that different implementations of services providing the same functionality can be chosen. Figure 5.5 shows that the MISS_COMB has

the smallest standard deviation and Figure 5.6 shows it has the smallest range, having the execution time with less variability.

5.9 Conclusions and future work

A Service-Oriented Architecture approach has been widely used in pervasive spaces such as smart homes. Nevertheless, we would like to enhance the process of composing services of heterogeneous SOAs. In this chapter, a framework is presented that supports automatic composition of heterogeneous SOAs services while requiring minimum intervention from the user. This framework provides for on-the-fly compositions, a searchable service directory, and support different SOAs with OSGi and WS as an example.

A case study is offered in which the performance of the composite service created by the framework is compared with composite services created using current techniques. The analysis of the results shows that the performance of our composition framework is acceptable and the resulting composite service show less variability than current composition techniques. The resulting service also performs close to the best-case scenario.

For future work, we would like to extend our composition framework by adding support for other SOAs, to compose a greater variety of services. We would also like to look at other measures to improve the composition framework performance and to automate the process of updating services. A study on how to incorporate formal methods that automatically check safety properties for the composite services is on going.

CHAPTER 6. A COMBINED MODEL CHECKING APPROACHFOR SAFETY OF COMPOSITE SERVICES

6.1 Abstract

Pervasive environments such as the smart home often rely on composite services to provide different functionalities. As the complexity of composite services increases, it becomes more important checking the safety of the interactions among different services and the workflow of the composites. Safety checking of composite services and their interactions is beneficial as some services might manage sensitive data or perform critical operations. In this chapter, we study the characteristics of services and categorize them as baseline or extended. A mechanism that relies on model checking techniques to ensure that services of both categories meet our adopted safety criteria is proposed. This mechanism is implemented as a fundamental part of an automatic composition framework to ensure that only composite services that meet the safety criteria are created. An example of a smart home composite service for medications management is provided to demonstrate our approach.

6.2 Introduction and Motivation

A smart home is a house that integrates different technologies to assist the elderly and persons with special needs to stay home longer and live more independently [1]. Smart homes can help the residents with their activities of daily living by monitoring and assessing conditions of the inhabitants. Services like those found in a smart home might perform critical operations or might manage sensitive data. This imposes several requirements, conditions, and properties over the services that define the safety criteria. Checking the safety of services implies checking compliance with the set of safety criteria. Therefore, a mechanism to guarantee that services satisfy the safety criteria as well as compliance with applicable laws is needed.

We provide a mechanism to ensure the safety of services such as those used in a smart home environment. To ensure the safety of the services our mechanism relies on the use formal software analysis. M.B. Dwyer et. al. [13] define formal software analysis as a

mathematically well-founded automated technique to reason about the behavior of a software system with respect to a specification of sound and unsound behavior. Model checking is one type of formal software analysis that has gained popularity as software becomes more complex and intractable. In our work, model checking techniques are integrated to verify whether composite services and their interactions satisfy the adopted safety criteria.

In the literature, different composition frameworks that use model checking techniques especially targeted for web services (WS) are found [112]. We compare their strategies with our solution. Several frameworks for automatic composition of WS select services based on particular requirements or QoS measures such as performance, response time, or reliability [3]. Another set of frameworks focus on formally modeling the individual services, use models of the services to create the composite service, and checking properties of the resulting models [113],[112]. Some of these frameworks embed the checking tasks within the composition process [114]. Our solution follows a similar strategy by modeling the selected services and their interactions, and checking safety properties on the model. However, in our solution, the interactions checked are among candidate services that belong to heterogeneous SOAs, and model checking of extended services is integrated as part of a composition framework and performed automatically. Research works on automatic composition frameworks of services based on heterogeneous architectures, either formally check the individual services or the resulting composite services [51]. In our work, we provide a safety checking solution that supports modeling and checking baseline and extended composite services of heterogeneous SOAs and their interactions. Our work describes the theoretical foundation of the model checking approach used, presents an actual implementation of the mechanism adopted, and incorporates it as part of our heterogeneous SOAs composition framework.

The service composition framework is extensible and allows automatic seamless composition of services of heterogeneous SOAs [115]. The goal of the composition framework is to have a mechanism that a user with little technical expertise can use to develop applications to control the devices and appliances in environments such as the smart home. In this chapter an enhancement to this composition framework is provided by adding

support for model checking, which will allow checking the safety criteria of individual services, composite services, and the safety of interactions among these. The safety criteria include the standard criteria and the custom criteria. The standard criteria are those properties that all services must comply. The custom criteria are custom properties that a particular composite service must comply. Services are divided based on their properties in two sets: baseline services and extended services. Baseline services are those that are assumed part of the initial configuration and may have custom safety criteria to be checked. Extended services are those that can be added, removed, started, or stopped at any time. We contribute with a detailed semi-automatic modeling and checking approach for baseline services. The extended services are dynamic and the details on how they operate may not be known a priori. To model and check those services we contribute with an automatic approach. Different strategies are used for modeling and checking services in different categories. For baseline services they are modeled in PROMELA, their custom properties are modeled using Linear Temporal Logic (LTL) and checked for compliance using the SPIN model checker as an example. Extended services are checked for safe interactions and compliance with safety properties such as controllability, absence of deadlocks, and absence of false nodes. We claim that the Simple Service Composition Language (SSCL), used to specify composite services workflow, is compatible with the Business Process Execution Language for Web Services (WS-BPEL), a widely used orchestration language. Composite services are formally modeled using open Workflow Net (oWFN) [116] [117] and checked using the Fiona model checker.

Furthermore, we use as an example to demonstrate the effectiveness of our solution a smart home composite service for the management of medications [68]. This is an appropriate example as it handles patient's personal and sensitive healthcare data including prescription details, health conditions, and other medicines the patient might be taking. Special care must be must taken to ensure safety and compliance with applicable laws such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States [86]. M.J. May et. al in [63] describes a representation of the law using formal language. This kind of formalization of the law is necessary as sometimes the language of the law can be

ambiguous or subject to interpretation of an expert such as an attorney or a physician. Such ambiguity makes it very difficult for computer systems to check whether a system complies with the law. We study several statements of the HIPAA law, formalize them, model them, and use them as our custom safety criteria.

6.3 Background

Several model checking approaches are used in this work for modeling and checking composite services and their interactions. Two model checkers, SPIN and Fiona are used. SPIN is a widely used model checker that interprets the model of a system specified in the Process Meta Language (PROMELA) and checks if the model satisfies properties that are specified using Linear Temporal Logic (LTL). The reader can refer to [84],[118] for more details on the SPIN model checker. The other model checker used is Fiona that is designed to interpret open Workflow Nets (oWFN) [119]. The motivation to use oWFN is that these structures are specially designed to model inter-organizational workflows and service-oriented architectures. They offer an efficient modeling structure as they reduce the number of states during analysis when compared to other structures such as canonical Petri Nets. Tools are available that generate oWFN from workflow specification files [120], [62]. Fiona takes as input an oWFN and checks a standard set of properties such as controllability, absence of cycles, and absence of false nodes. This section presents background details on the theory of oWFN by first showing the definition of a Petri Net, followed by the definition of the oWFN and then a definition of a controller that is an automaton used to determine controllability.

Definition 1. A *Petri net* N consists of:

- A set P of places, usually represented as a circle
- A set T of transitions, usually represented as a rectangle
- A flow relation F, where $F \subseteq (T \times P) \cup (P \times T)$, represented by edges
- A marking m that is a multiset $m: P \rightarrow N$ (where $m[p]$, represents the number of tokens in place p). A token is usually represented by a dot.

- A marking *m* enables a transition *t* if for each place *p* with $(p, t) \in F$, $m[p] \geq 1$. If enabled at *m*, firing *t* yields the marking *m'* with $m'[p] = m[p] - 1$ if $(p, t) \in F$ and $(t, p) \notin F$, $m'[p] = m[p] + 1$ if $(t, p) \in F$ and $(p, t) \notin F$, and $m'[p] = m[p]$ otherwise.

Figure 6.1 shows a graphical representation of a Petri Net. This figure illustrates the different markings and the firing process.

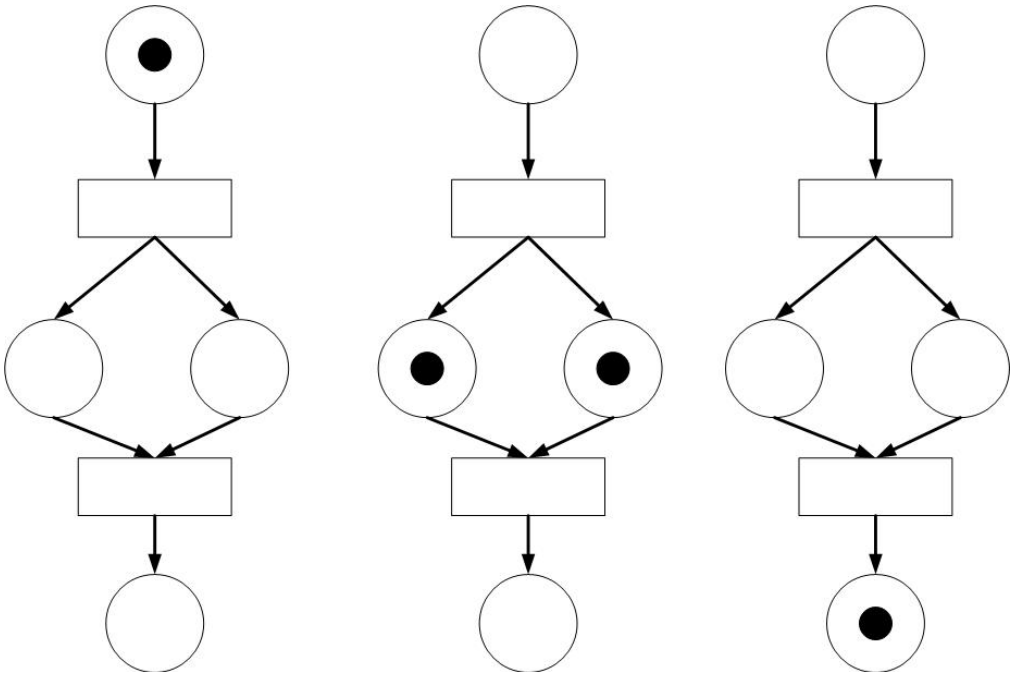

Figure 26 - Graphical representation of a Petri Net and its firing.

Definition 2. A Petri net *N* is an *open Workflow Net (oWFN) M* if:

- *P* is the disjoint union of the sets P_m, P_i and P_o, where P_m is the set of internal places as defined *Definition 1* for Petri Nets, P_i is the set of input places with no incoming edges, and P_o is the set the output places with no outgoing edges.
- $F \cap (P_o \times T) = \emptyset$
- $F \cap (T \times P_i) = \emptyset$

- *F* does not contain cycles (the transitive closure of *F* is irreflexive)
- *M* has a distinguished initial marking m_0
- *M* has a set Ω of distinguished final markings

Figure 6.2 shows a graphical representation of an oWFN. Initial marking only have outgoing edges. Input places and transitions from input places are colored orange. Output places and transitions to output places are colored yellow. Final markings have only incoming edges and are colored gray.

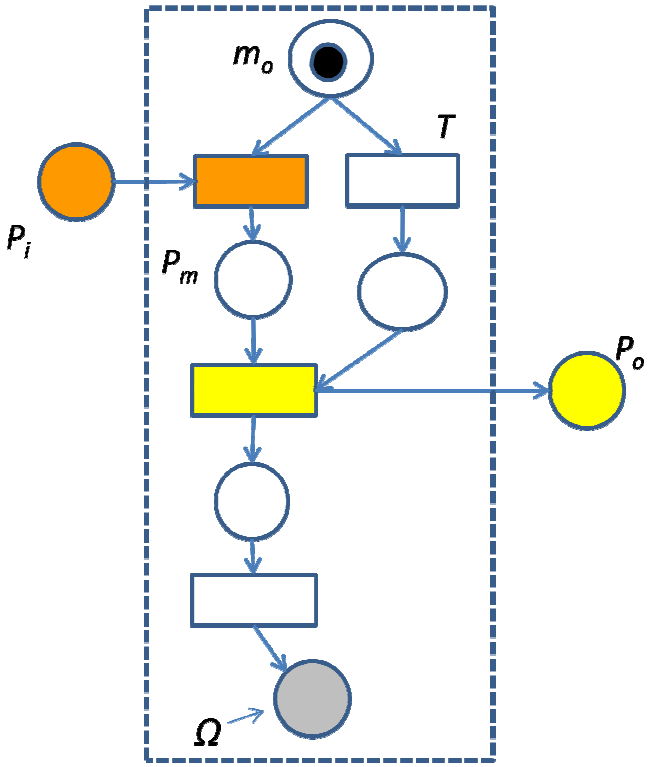

Figure 27 - Graphical representation of an oWFN

Definition 3. Let *M* be an oWFN and $I \subseteq (P_I \cup P_O)$. A *controller* is an automaton connected to *I* containing the following items:

- Alphabet *bags(I)*

- A set of states Q
- A move relation $\delta : Q \times bags(I) -> \rho(Q)$
- And an initial state q_0

Some of the safety properties that can be modeled and verified with oWFN are *weak soundness, soundness, usability, controllability, absence of cycles* and absence of false nodes. *Weak soundness* determines the possibility to reach an end state from each state reachable from the initial state. *Soundness* means in addition to *weak soundness* that there are also no dead transitions on the net, where dead transitions are those that cannot fire. *Usability* indicates that there exists an environment such that the oWFN of a service composed with the oWFN of the environment yields a weakly sound net. *Controllability* occurs when given a partner service for an oWFN, a *controller* for the composition can be constructed and *soundness* holds. *Absence of cycles* means that given a partner for and oWFN the composition does not create any cycle. A*bsence of false nodes* checks that given a partner for and oWFN, the input places matches the annotations of the corresponding output places. For our work, we are particularly interested in checking for *controllability, absence of cycles,* and *absence of false nodes* as our safety criteria.

There are several techniques for checking *controllability* such as computing a public view (*PV*), an interaction graph (*IG*) or an operating guideline (*OG*) [117]. A *PV* is an abstract representation of the operations and communication behavior of a service that describes the publicly available service interface. An *IG* is a structure that represents the controller point of view of a node, where each node contains all possible states that are reachable at certain point. An *IG* can be seen as a hypothesis for the controller, representing feasible runs of a partner service. An *OG* is a structure that represents the behaviors of all possible strategies for sound executions. It describes how a partner service should behave instead of describing the actual service behavior. Computationally speaking, out of the three methods, constructing the *PV* is the least costly, but the controllability checks are more expensive as *PV* greatly suffers the state explosion problem. Another issue with *PV* is that it reveals too much information about the service operation. The *IG* is relatively easy to compute and to verify,

but it also reveals too much information about the possible states of the service. The *OG* is the hardest to compute but it is easy to verify. *OG* offers the advantage that it does not reveal information about the service itself as it describes how a partner should behave instead. Researchers generally prefers *OG* because, even though computing it is computationally more expensive, this is a onetime cost justifiable by the efficient verification and the fact that *OG* does not reveal information about the service but about a partner [121].

6.4 Model Checking Composite Services

Our main goal is to provide a mechanism to check the safety of composite services and their interactions, using as an example services encountered in smart home environments. The set of common services studied are classified as:

- *Baseline services*: Consists of those services that are assumed to be part of the initial configuration and may have custom safety criteria to be checked. (e.g. Notifications service and appliances control service)

- *Extended services*: Consist of services that can be added, removed, started, or stopped at any time. (e.g. monitor a special medical treatment for a month)

Although there are many approaches for formally checking services and applications, these solutions usually check services individually [13]. In our work, different approaches are combined to ensure that the services running in the system comply with our specific safety criteria. For baseline services, a semi-automatic approach that models services using PROMELA, specifies its custom safety criteria using LTL, and checks the standard and the custom safety criteria using the SPIN model checker is showed. For extended services in a dynamic environment, automatically checking the safety of these services interactions presents a challenge. The extended composite services are modeled as oWFN and the standard safety criteria are checked using Fiona model checker. We provide an example of our approach applicable to smart home environments. The example used is based on the Medicine Information Support System (MISS) [68],[111] that is a medication management

service. MISS is composed of several services, which are modeled together with their interactions and the safety criteria.

Figure 6.3 presents the architecture on how baseline services are modeled and checked. At the top, the *requirements* box, indicates all the requirements for a particular composite service. Based on these requirements the composite service is designed. Based on the *design*, a model of the composite service can be created using PROMELA. Based on the safety requirements, the custom *safety criteria* can be modeled using LTL. Providing the PROMELA model and the LTL model, the SPIN model checker is used to verify whether the model complies with the safety criteria specified. If the model do not satisfies it, goes back to the *design* for revision of the model. If the model satisfies the safety criteria, it passes to the implementation stage and finally the service is installed into the system, such as the *Smart Home OSGi Framework*.

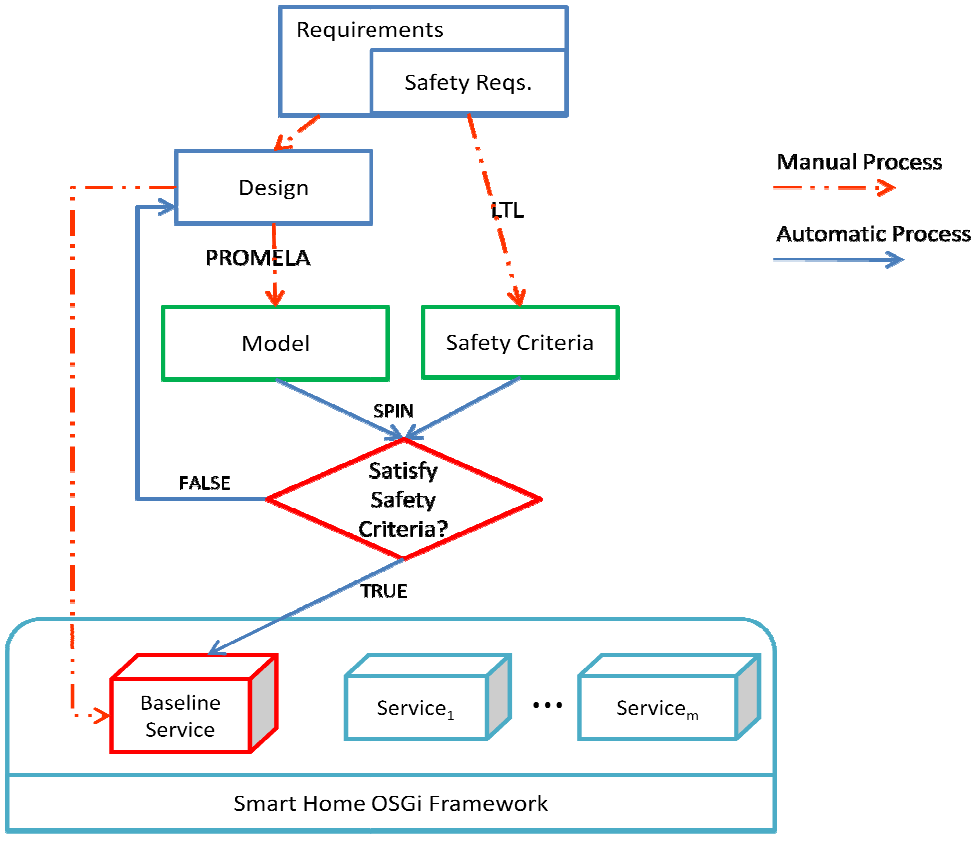

Figure 28 - Model Checking Architecture for Baseline and Extended Services

Figure 6.4 presents the architecture on how extended services are modeled and checked. At the top, the *requirements* box, indicates the requirements for a particular composite service. Based on these requirements the composite service workflow and their interactions are specified using SSCL. The SSCL file is then modeled as an oWFN automatically. This oWFN is then passed to the Fiona model checker that will check controllability, absence of cycles, and false nodes of the composite service. When an error is found, an error message is returned for corrections to the workflow. If no errors are found, the composition framework proceeds to generate automatically the implementation of the service. Finally, the composite service implementation is deployed into the environment, such as the *Smart Home OSGi Framework.*

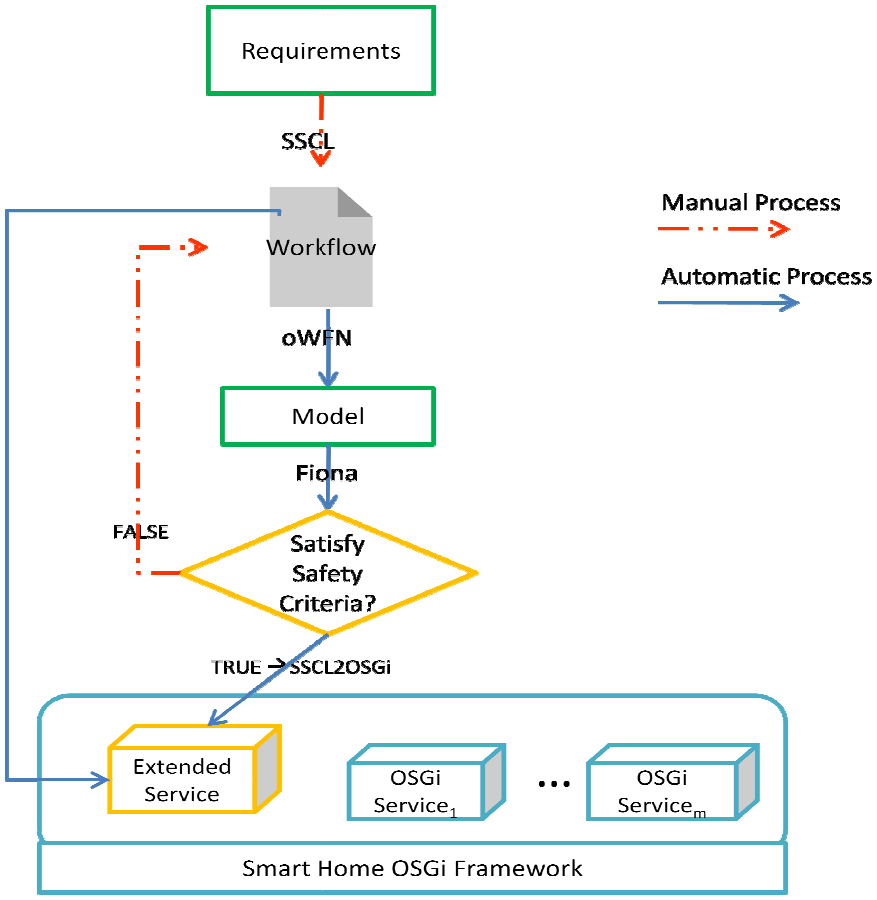

Figure 294 - Model Checking Architecture for Extended Services

6.5 Modeling and Checking Approach for Baseline Services

Baseline services are fundamental services in a smart home that form part of its core infrastructure and are expected to operate continuously. As these services are essential for the proper functioning of the smart home, special care should be taken in checking their proper functionality and compliance with safety requirements and applicable laws. We assume that the set of baseline services are defined at design time, which gives us a clear idea how they operate. We also assume that these services have a set of custom safety criteria they have to comply with. We show how to integrate safety checking into our composition framework by using the PROMELA language to model baseline services and Linear Temporal Logic (LTL) to model the custom safety criteria. For verifying compliance of services with the safety custom criteria the SPIN model checker is used [67]. In the next sections, more details on an example how to model of a system and the custom safety properties to check are provided.

6.5.1 Example of Services Modeled Using PROMELA

Our approach to model baseline services is shown using as an example the Medicine Information Support System (MISS), which integrates the doctor, the pharmacy, and the smart home to help the patient manage medications [111]. MISS ensures safety by checking possible drug interactions among medications, foods, and health conditions. The doctor, pharmacy, and smart home entities of MISS are modeled using PROMELA. Also the medicine conflict database (MCD) and a client record request are modeled. The custom safety criteria to check are based on the Health Insurance Portability and Accountability Act (HIPAA) in the United States [86]. A set of applicable HIPAA statements are modeled using LTL. In the models, covered entities that can be the patient, the government, or another entity that handles health records are included. The actual checking of whether MISS complies with the HIPAA law is performed using the SPIN model checker. The description on how each subsystem is modeled is next.

For the case of the doctor service there are two possible scenarios categorized as a regular flow and an alternate flow. The regular flow models a regular doctor visit where the doctor checks a patient and prescribes some medication. The alternative flow models when another

party such as a healthcare provider requests a patient record. A high level description of the doctor's service model is the following:

Doctor's Service:

Regular Flow:

- Assigns values to the variables of the following records:
    - Covered-entity
    - Local patient record
- Receives a patient
- Creates a prescription for the patient
- Checks for conflicts using a channel to communicate with the MCD process
- Forwards the prescription to the pharmacy service using a channel

Alternative Flow:

- Receives a request for a patient's record
- Serves the request for a patient's record depending on the requestor

The pharmacy service is modeled similarly by determining a regular flow and an alternate flow. The regular flow models when a pharmacist receives a prescription from the doctor, prepares it, and serves it. The alternative flow models when another party requests a patient record. The high-level view of the pharmacy's model is the following:

Pharmacy Service:

Regular Flow:

- Assigns values to the variables of the following records:
    - Covered-entity
    - Local patient record
- Receives the prescription data over a channel
- Checks for conflicts using a channel to communicate with the MCD process
- Forwards the prescription to the smart home using a channel

Alternative Flow:

- Receives a request for a patient's record
- Serves the request for a patient's record depending on the requestor

The smart home service is also modeled with a regular flow and an alternate flow. The regular flow models when the smart home receives a prescription from the pharmacy, checks for conflicts with medications and foods at home, and checks for timeliness and completeness as described in [111]. The alternative flow models when another party requests a patient's record. The high-level view of our smart homes model is the following:

Smart home Service:

Regular Flow:

- Assigns values to the variables of the following records:
    - Covered-entity

- Local patient record

- Receives the prescription over a channel
- Checks for conflicts using a channel to communicate with the MCD process
- Determines completeness and timeliness

Alternative Flow:

- Receives a request for a patient's record
- Serves the request for a patient's record depending on the requestor

The doctor, pharmacy, and smart home services check for different conflicts among medications, foods and medical conditions using the medicine conflict database (MCD) modeled as follows:

MCD Service:

- Receives the new prescription
- Receives the patient record
- Determines if there is a conflict based on the prescription and patient record

A person may request a patient's record from the doctor, the pharmacy, or the smart home. This client can be the patient, the government, or any other covered entity [86]. Client request are modeled as follows:

Client Request:

- Determines the category of the requestor: patient, government or covered entity
- Request a patient's record to one of the subsystem: doctor, pharmacy or smart home

- Returns the corresponding patient's record

With the doctor, pharmacy, smart home, MCD, and client request models, the overall functionality of MISS is captured. The model of the system in PROMELA can be found in APPENDIX C. The details on how to model the safety criteria to check on these models is provided next.

6.5.2 Safety Criteria in Compliance with the HIPAA Law

As smart homes aim to assist persons with their activities of daily living including healthcare, it is important checking how a system like MISS manages the sensitive data of healthcare records. We provide an example that checks whether the MISS model complies with custom safety criteria based on healthcare law. The law modeled is the HIPAA law, which is collection of requirements for the management of healthcare data that covers healthcare related entities. The law is modeled using a mapping of the legal language into formal language understandable by a computer system [63]. The statements of the law modeled are those that we believe directly apply to smart homes and to our MISS example. The interpretation of these statements of the law and their model in linear temporal logic (LTL) formulas are provided. These LTL formulas are used as our custom safety criteria. The text of the HIPAA law can be found in [86].

Statement 1

Interpretation: If a covered entity requests a patient's record, it will never receive an empty record.

LTL Formula:

#define p_1 *(requestor == 2)*

#define q_1 *(patientData == true || restrictedData == true)*

$[]\ ((p_1) \rightarrow (<>\ (q_1)))$

Explanation: The *requestor* variable indicates the person or entity requesting the data and the value 2 represents a covered entity. The *patientData* and *restrictedData* variables represent the two components of a patient's record.

Statement 2

Interpretation: If the MCD finds a conflict, it is never the case that the calling process reaches the end.

LTL Formula:

#define p_2 *(response == true)*

#define q_2 *(doctorFinished == true || pharmacyFinished == true || shFinished == true)*

[] ((p_2*) -> (<> (!*q_2*)))*

Explanation: The *response* variable indicates the result of the conflict checking returned by MCD. The *doctorFinished*, *pharmacyFinished*, and *shFinished* variables indicate whether a service finishes its execution.

Statement 3

Interpretation: If the requestor is the government it is always the case that the full record will eventually be disclosed

LTL Formula:

#define p_3 *(requestor == 3)*

#define q (patientData == true && restrictedData == true)

[] ((p_3*) -> (<> (*q_3*)))*

Explanation: The *requestor* variable indicates the person or entity requesting the data with the value 3 representing the government. The *patientData* and *restrictedData* variables represent the two components of a patient's record.

Statement 4

Interpretation: If timeliness, completeness becomes false or response becomes true, then eventually the system gives a notification.

LTL Formula:

#define p_4 *(timeliness == true || completeness == true || response == true)*

#define q_4 *(notification)*

$[] ((p_4) \rightarrow (<> (q_4)))$

Explanation: The *timeliness* variable indicates if the medicine is taken at the correct time. The *completeness* variable indicates if the dosage of medicine taken is correct. The *response* variable indicates if there is a conflict. The *notification* variable indicates if a notification is given.

Statement 5

Interpretation: When patients request their medical record, only the patient's data is disclosed but never the restricted data.

LTL Formula:

#define p_5 *(requestor == 1)*

#define q_5 *(patientData == true && restrictedData == false)*

$[] ((p_5) \rightarrow (<> (q_5)))$

Explanation: The *requestor* variable indicates the person or entity requesting the data with the value 1 representing the patient. The *patientData* and *restrictedData* variables represent the two components of a patient's record.

Statement 6

Interpretation: The doctor has to satisfy all the law requirements for covered entities over time.

LTL Formula:

#define p_6 *(doctorCE.patientReqs == true && doctorCE.adminReqs == true)*

<>[] (p_6)

Explanation: The *doctorCE.patientReqs* indicates if the doctor is fulfilling the requirements for patients. The *doctorCE.adminReqs* indicates if the doctor is fulfilling the administrative requirements.

Statement 7

Interpretation: The pharmacy has to comply with the patient and administrative requirements over time.

LTL Formula:

#define p_7 *(pharmacyCE.patientReqs == true && pharmacyCE.adminReqs == true)*

<> [] (p_7)

Explanation: The *pharmacyCE.patientReqs* indicates if the pharmacy is fulfilling the requirements for patients. The *pharmacyCE.adminReqs* indicates if the pharmacy is fulfilling the administrative requirements.

Statement 8

Interpretation: The smart home has to fulfill the patient and administrative requirements of HIPAA over time.

LTL Formula:

#define p_8 *(shCE.patientReqs == true && shCE.adminReqs == true)*

<> [] (p_8)

Explanation: The *shCE.patientReqs* indicates if the smart home is fulfilling the patient requirements. The *shCE.adminReqs* indicates if the smart home is fulfilling the administrative requirements.

The PROMELA model of MISS and the LTL model of the HIPPA law show a particular example on how to model baseline services and their safety criteria. However, any appropriate model checking tools can be used for this purpose.

6.6 Modeling and Checking Approach for Extended Services

In environments such as the smart home, it is desired for services to comply with a standard set of safety criteria. Extended services automatically created by the composition framework should also comply with this criteria and be formally checked. In this section, we show how the composite service workflow specified in SSCL is converted into an oWFN. Several syntactic and semantic properties of the workflow specification are defined and proved. It is also shown how the model checker is integrated into the composition framework and how these checks are automated.

6.6.1 The SSCL2OSGi Composition Framework Description

The SSCL2OSGi composition framework supports automatic composition of services of heterogeneous SOAs [115]. This chapter enhances the framework by adding formal modeling and checking of services and their interactions. The SSCL2OSGi composition framework accepts as input a composite service workflow description specified using the Simple Service Composition Language (SSCL). The composition framework parses and analyzes the SSCL file for creating a composite service that relies on services of heterogeneous SOAs. The description and binding information of each service are stored in a repository that supports registering heterogeneous SOA services. The composition framework produces the implementation of the workflow as an OSGi service. The SSCL2OSGi framework is

developed using OSGi technology and currently supports WS and OSGi services. The safety of the composite service and their interactions as specified in the SSCL file are verified before creating the composite service implementation.

6.6.2 Converting SSCL to oWFN

This section explains the approach to model a service workflow specified in SSCL using open Work Flow Nets (oWFN). Several works have tackled similar problems and produced tools such as those for converting the Business Process Execution Language for Web Services (WS-BPEL) into Petri Nets [122],[123]. Other works present tools for converting WS-BPEL into an oWFN [120],[61]. In this work, a similar approach is used of converting a SSCL workflow file into an oWFN by reducing it into WS-BPEL. This will allow using those tools already available for verifying oWFN.

We provide a set of definitions and prove a set of claims to show how to convert a SSCL files into an oWFN. The SSCL is a language inspired by WS-BPEL. The SSCL language is used as it provides a SOA-independent language to specify composites. WS-BPEL is a WS language and its syntax is targeted to that specific architecture, while SSCL does not target any specific one. SSCL is designed based WS-BPEL syntax and the conversion algorithm used by N. Lohmann [120] to convert WS-BPEL files into oWFN. We claim that SSCL is semantically compatible with WS-BPEL. We also claim that the SSCL design allows it to be converted into oWFN. Finally, we show that an oWFN produced from a SSCL file and an oWFN produced from a compatible WS-BPEL file are the same. Now, we provide a series of definitions that will be used to prove our claims:

Definition 4. Formal Syntax of SSCL

Please refer to Appendix A □

Definition 5. *Semantic tags*

A *semantic tag* t_s is a tag or attribute that carries semantic meaning and is used by the conversion algorithm. (e.g. *if* and *invoke*) □

Definition 6. *Syntactic tags*

A *syntactic tag* t_i is a tag or attribute that carries only syntax but does not have semantic meaning. These tags are optional and ignored by the conversion algorithm. (e.g. *name* and *role*) □

Definition 7. *Compatibility* of a SSCL and a WS-BPEL command

An SSCL command c_s is *compatible* with a WS-BPEL command c_b if the semantic tags of c_s have the same semantic meaning as the semantic tags of c_b but their syntactic tags may differ. We denote compatibility of commands as $c_s \sim c_b$ □

Definition 8. *Compatibility* of SSCL and BPEL processes

A process *s* described in SSCL is *compatible* with a process *b* described in WS-BPEL if all their commands are *compatible* and they appear in the same order. We denote compatibility of processes as $s \sim b$ □

Claim 1. The *if* command in SSCL if_s is *compatible* with the *if* command in WS-BPEL if_b and both commands map into the same oWFN structure.

Proof. We first show that if_s is *compatible* with if_b. Both commands evaluate a Boolean condition and based on the result of the evaluation a branch is executed. Both commands support *elseif* and *else*. Both commands have the same set of *semantic tags*, therefore $if_s \sim if_b$.

The differences between if_s and if_b are a set of *syntactic tags*. The conversion algorithm ignores these *syntactic tags*, therefore both commands will map into the same oWFN structure □

We can use a similar argument as in *Claim 1* for showing the compatibility of the commands *assign, repeatUntil, forEach, while,* and *invoke*. The rest of the tags and attributes supported by SSCL are not used by the algorithm that creates oWFN structures. They are

syntactic tags that lack semantic meaning. Now we proceed to prove the claims about the produced oWFNs.

Claim 2. An SSCL process p_s produces the same oWFN as its *compatible* WS-BPEL process p_b.

Proof. Given p_s, its *compatible* process p_b have the same *semantic tags* in the same order according to Definition 8.

Claim 1 shows that the *if* commands will map into the same oWFN structure

Using the same strategy of *Claim 1* will show the compatibility of the commands *assign, repeatUntil, forEach, while,* and *invoke.*

Therefore, the oWFN produced for a p_s process is the same as for its compatible p_b □

After proving the claim that the oWFN generated from a SSCL files is the same as the one generated from a *compatible* WS-BPEL processes, now the tools for analyzing and checking properties of oWFN can be used [124]. APPENDIX D has the oWFN model of the MISS system and figure 6.5 shows a graph of the simplified oWFN, which is the same oWFN we obtain from the *compatible* WS-BPEL file**.** The next section contains details on how to check safety criteria on this oWFN using the Fiona model checker.

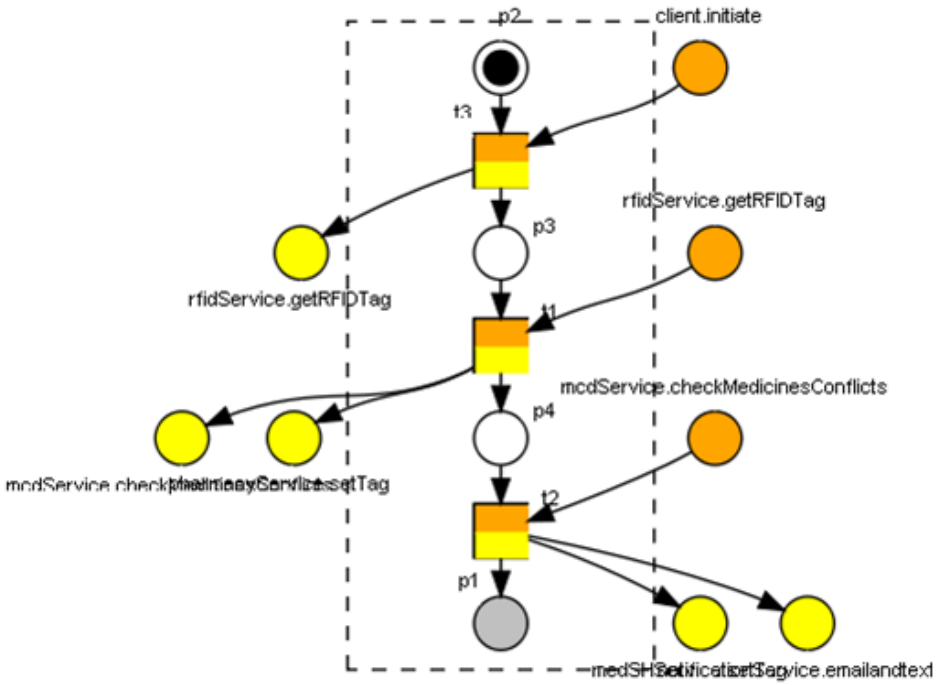

Figure 305 - Structurally reduced oWFN generated from the SSCL process

6.6.3 Model Checking oWFN Using Fiona

Fiona [62] is a model checking tool that analyzes the safety of interactions among services modeled as oWFN [61]. Fiona can determine matching services, verify partner synthesis and perform a check for controllability, which is a minimal correctness criterion stating the existence of a behavioral compatible partner for the service. Fiona also provides the tools to create an adapter rule that serves as a mediator between different oWFN, mapping items from one oWFN into items in another. It can also check for absence of cycles and absence of false nodes. Fiona can produce the public view (*PV*), interaction graph (*IG*), and operating guideline (*OG*) of composite services automatically.

For the purpose of our work, Fiona model checker is used for automatically checking the oWFN models generated from the SSCL files. Compliance with a standard set of safety

requirements are checked by integrating Fiona with the SSCL2OSGi composition framework and making model checking a necessary step before creating the composite service implementation. Different composite services were tested to ensure satisfaction of our safety criteria that consist of checking for controllability, absence of cycles, and absence of false nodes. The results obtained by checking MISS controllability using *PV*, *IG*, and *OG* are presented next.

6.6.4 Computing Public View (PV):

A *PV* is an abstract version of a service and its communication behavior that describes the publicly available service interface and the different possible ways to interact with it. To check controllability a new service *R* is composed with the *PV* of another service and Fiona checks the composite for controllability. Computationally speaking, constructing the *PV* is faster than computing the *IG* or the *OG*. However, creating the composite service and checking for controllability is computationally more expensive as public view suffers from the state explosion problem. Figure 6.6 shows a partial view of the *PV* graph of the MISS system.

6.6.5 Computing Interaction Graph (IG):

An *IG* represents the controller point of view of a node, containing all possible states that are reachable at each point during execution. The *IG* describes a hypothesis for the controller, representing feasible runs of a partner service according to the interaction protocol. The *IG* can be used to determine controllability by looking for a path from the initial node to the end node. Figure 6.7 shows the *IG* of the MISS system.

6.6.6 Computing Operating Guideline (OG):

An *OG* represents a description of the behaviors of all strategies for *sound* executions. It describes the expected behavior of a partner service. This way a partner service can verify if its behavior matches the expected behavior for partner services. The *OG* can verify controllability by finding a path from the initial node to an end node, which will determine if the partner service matches the *OG* specification. Figure 6.8 shows the *OG* of the MISS system.

6.6.7 Check for Cycles, False Nodes and Controllability

Fiona is used also to check the absence of cycles and absence of false nodes. Checking for cycles avoids infinite calls to services. Checking false nodes helps to identify nodes violating their own annotation. For computing controllability, we primarily use *OG*, provide support for *IG*, but do not support *PV* in our composition framework.

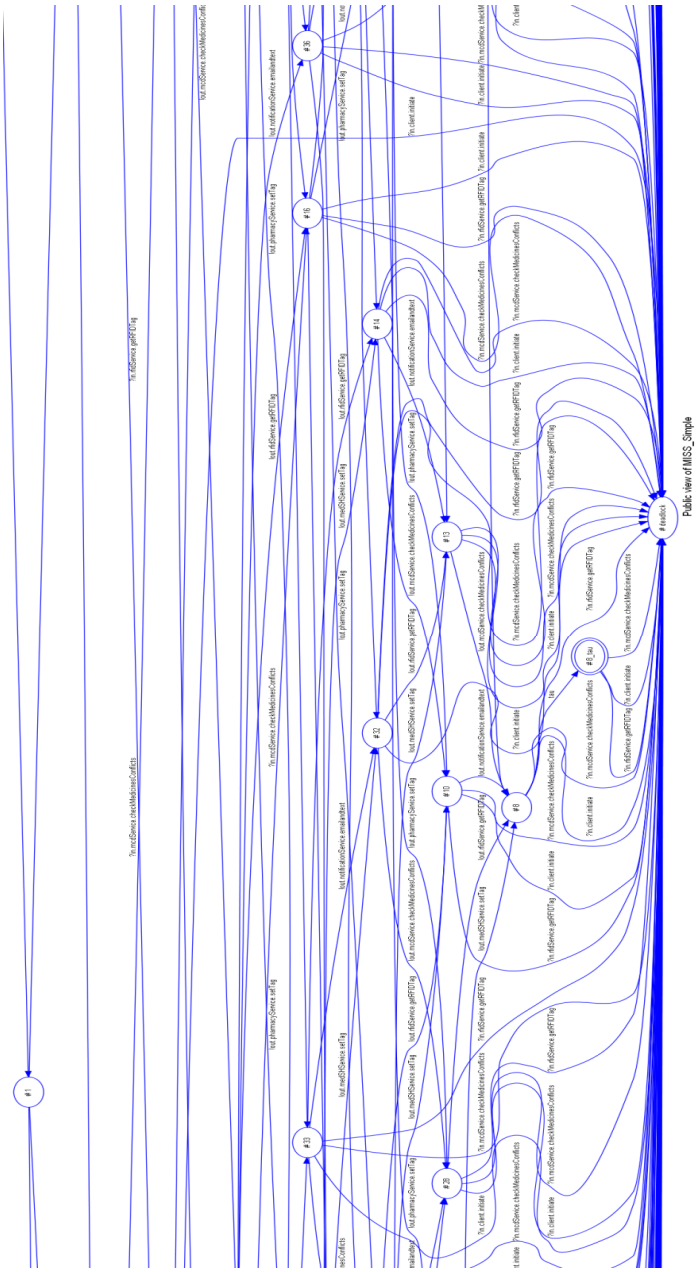

Figure 316 - Partial Public View Graph of the MISS system

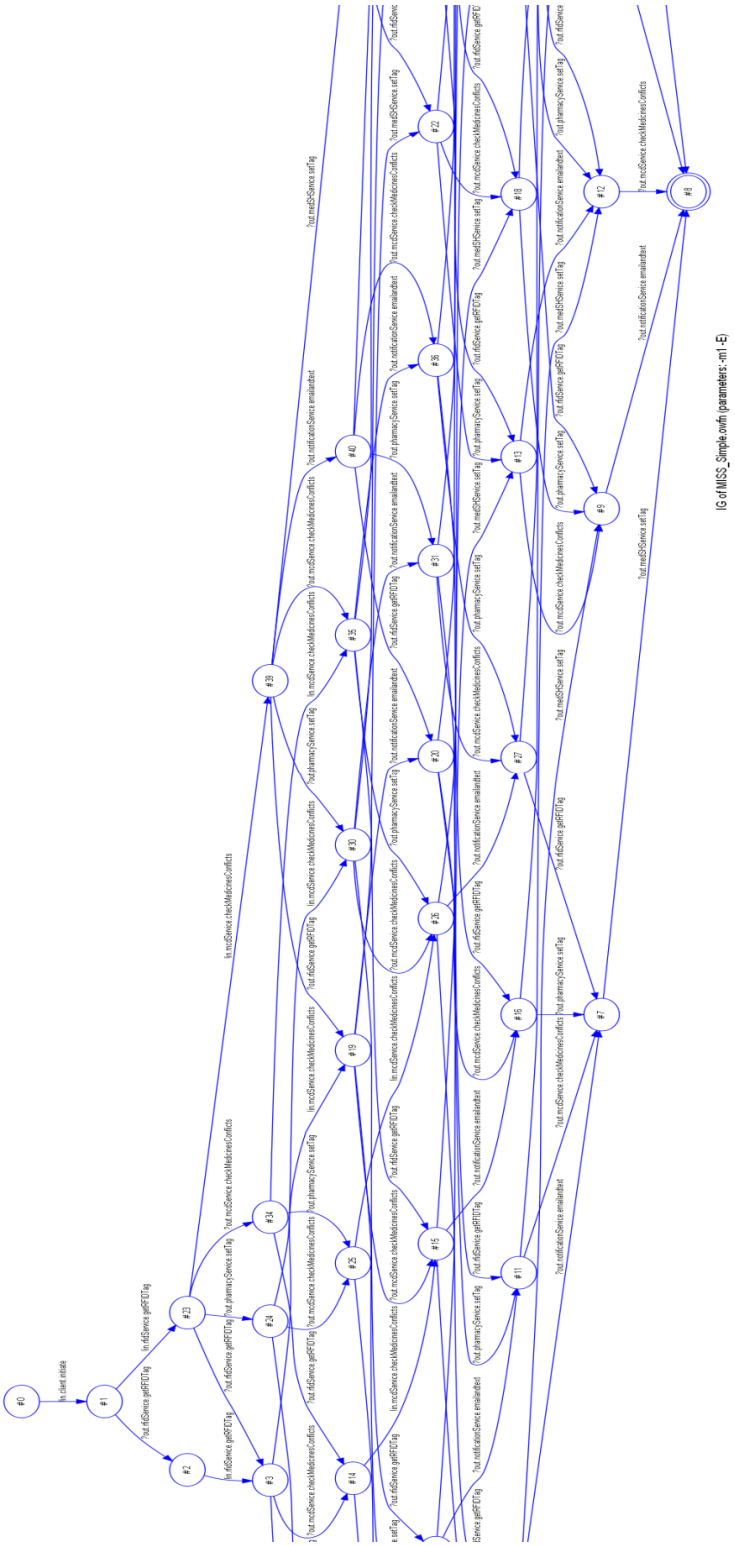

Figure 327 - Partial Interaction Graph of the MISS system

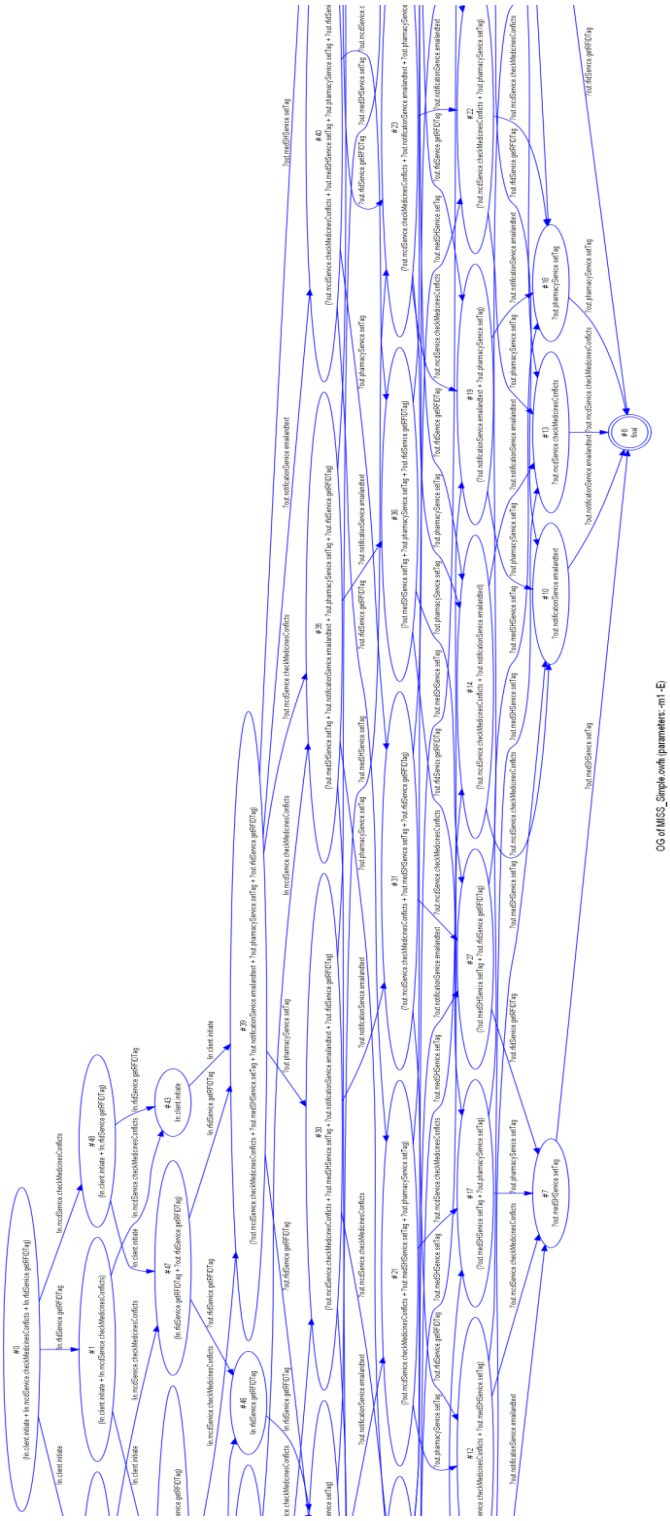

Figure 338 - Partial Operating Guideline Graph of the MISS System

6.7 Composite Services Safety Proofs

In this section, it is shown that by using these model checking approaches, baseline and extended services are safe before, during, and after composition.

Claim 3. A baseline service *s* is *safe*, meaning it respects custom safety criteria *c*, before composition

First, we show baseline service *s* to be *safe* with respect to custom safety criteria *c*. It has been shown that the SPIN model checker determines whether a PROMELA model satisfies LTL properties [84]. We already modeled baseline services using PROMELA and the safety criteria based on the HIPAA law using LTL. Let this model be *s* and the safety criteria be *c*. Only those services *s* that satisfies safety criteria *c* based on the results obtained by the model checker SPIN are used, therefore showing a baseline service *s* to be *safe* before composition □

Claim 4. The extended service interactions among *safe* baseline services $s_1, \ldots, s_n$ with respect to their safety criteria $c_1, \ldots, c_m$ are also *safe*.

Proof. Individual baseline services are shown to be *safe* in *Claim 3*. Now we prove that the interactions among these services are also *safe* with respect to our safety criteria. We showed previously how to generate an oWFN from a SSCL process. It has also been shown that the Fiona model checker can determine the safety of interacting services [62],[125]. Only those services that the Fiona model checker determines that satisfy the safety criteria are accepted, therefore the extended services interactions among *safe* baseline services is *safe* □

Claim 5. The services running within the smart home are *safe* before, during, and after the composition.

Proof.

Claim 3 shows that baseline services are *safe* with respect to their safety criteria.

Claim 4 shows that the interactions among *safe* baseline services are safe with respect to their safety criteria.

Services running in the environment before composition are the baseline services that *Claim 3* shows are *safe*. During the composition, *Claim 4* shows that these services are checked ensuring safety during composition. These model checking techniques are incorporated into our automatic composition framework. The framework only accepts those composite services that pass all the safety tests. Therefore, composite services are *safe* after the composition. This shows that services running within the environment are *safe* before, during, and after the composition □

6.8 Conclusions and future work

We present a model checking approach to ensure the safety of composite services. Services are divided into the categories baseline and extended. Baseline services are assumed to be always up and running, while extended services are more dynamic and may come and go. An example based on the Medicine Information Support System for smart homes is provided where standard safety criteria for both types of services are checked and custom safety criteria is checked for baseline services.

The model checking approach for extended services is integrated with an automatic composition framework supporting services of heterogeneous SOAs. This formal check allows implementing only services proven safe. This way the safety criteria for services are satisfied before, during and after compositions. Examples on how to model and check these services are provided using the tools SPIN for baseline services and Fiona for extended services.

This model checking approach for services in environments such as the smart home provides a stronger argument when claiming the safety of our system. This is especially important for composite services such as those found in smart home environments as they might perform critical operation or handle sensitive data. Our approach and integration into an automatic composition framework ensures that services are continuously fulfilling the

safety criteria and that only new composite services that meet the safety criteria are accepted. This work contributes in providing a mechanism for having safe, reliable services of heterogeneous SOAs using formal methods. In addition, we formally check whether composite services comply with applicable custom safety criteria such as healthcare laws. This certainly sets a higher standard for SOA development by making the use of formal methods a fundamental part of the composition process for service.

CHAPTER 7. COMPOSITION FRAMEWORK PERFORMANCE EVALUATION AND ANALYSIS

7.1 Introduction

This chapter presents a performance analysis of our composition framework. A series of hypothesis are introduced, several experiments conducted under different networking environments, the data is collected and analyzed, and a summary on the results obtained is presented. Throughout this work, the details of our composition framework for services of heterogeneous SOAs have been described and how model checking techniques were incorporated to ensure the safety of composite services and their interactions. Now we present an analysis of the performance of the composition framework using as an example smart home composites services.

The examples used to test the composition framework are based on the Medicine Information Support System (MISS) described in chapter 4. Three different versions of the MISS composite service are used in the experiments. One of composite service only uses OSGi services and it is identified as MISS_OSGi. Other composite service only uses WS and it is named MISS_WS. The last composite service is automatically generated by the composition framework. It combines OSGi and WS and it is called MISS_COMB. The execution times of these three composite services are measured under a regular and a constrained networking environment. For the regular network an Ethernet LAN is used while for the constrained a wireless mesh network (WMN). Composite services were tested under these two networking environments as services might be accessed from locations where resource might be limited such as smart homes deployed rural areas. A regular LAN network represents what generally is found under standard networking conditions. The WMNs have resource constraints and are less reliability, representing networking conditions where resources might be more limited. Several experiments were designed and conducted based on the following hypothesis:

Hypothesis 1. A regular LAN network performs better than a WMN

Hypothesis 2. Given the three implementations of the same composite service: MISS_OSGi, MISS_WS and MISS_COMB; MISS_OSGi will have the best execution time performance, followed by MISS_COMB, and MISS_WS will have the worst execution time performance.

Hypothesis 3. Given three implementations of the same composite service: MISS_OSGi, MISS_WS and MISS_COMB; MISS_COMB will show less variability, followed by MISS_OSGi and finally the MISS_WS will have more variability.

Hypothesis 4. A WMN has more variability that a regular LAN network.

7.2 Experiment Setup

We set up two environments to test the composition framework. The first environment is regular Ethernet LAN. The second environment is a WMN. The details of each experiment setup are provided next.

7.2.1 Regular Network Setup

For the experiments under a regular network, the server used is a Dell Dimension 9200 with Windows Vista as its operating system. The server provides supports for OSGi and WS. The OSGi framework used is knopflerfish version 2.2.0. This machine also runs a customized version of Apache jUDDI UDDI server with Apache Tomcat 5.5 as the HTTP server.

The client machine is a Dell Optiplex GX280 PC with Windows XP as its operating system. The OSGi framework used is knopflerfish version 2.2.0 with the automatic composition framework already installed.

These machines are under the Ethernet LAN of the Department of Computer Science at Iowa State University, which provides the regular networking environment. A set of four experiments are conducted. One experiment measures the time taken by our automatic composition framework to create a composite service of MISS that combines OSGi and WS. Another experiment measures the execution time of MISS_OSGi. Another experiment tracks

the execution time of MISS_WS. Finally, the execution time of MISS_COMB is measured. Each experiment is repeated at least a 100 times.

7.2.2 Wireless Mesh Network Setup

For the experiments under the WMN, the server is a Dell Optiplex GX280 with Windows XP as its operating system. This machine has two wireless interfaces one used for upstream another for downstream. The server provides supports for OSGi and WS. The OSGi framework used is knopflerfish version 2.2.0. This machine also runs the customized Apache jUDDI UDDI server with Apache Tomcat 5.5 as the HTTP server.

The client machine is a Dell Optiplex GX280 PC with Windows XP as its operating system. This machine also has two wireless interfaces one for upstream the other for downstream. The OSGi framework used in this machine is knopflerfish version 2.2.0 with the composition framework already installed.

To set up the WMN network infrastructure experiments, four computers are used: the server, the client, and two machines in the middle that forward packets between the client and the server. The Microsoft Research Wireless Mesh Toolkit [126] is installed in all the mesh machines to support wireless mesh network connectivity. Under the WMN, the same four experiments as in the regular network are conducted measuring their execution time. An experiment that measures the execution time of MISS_OSGi, another that tracks MISS_WS, the time taken by our automatic composition framework to create a composite service and the execution time of MISS_COMB. Each experiment is repeated at least a 100 times.

7.3 Overhead of the Composition Framework

This section shows the results of the time taken by the composition framework to create the composite service that implements MISS. The execution time of the following activities is measured: parsing and binding, model check, code generation, and creating the bundle. The parsing and binding activity includes parsing the SSCL file, querying the UDDI registry, getting the services binding information and binding to the services. The model checking activity includes the creation of the oWFN and performing the services interactions checks using the Fiona model checker as described in Chapter 6. The code generation activity

involves translating the SSCL orchestration code into executable Java code, the creation of the service interface, the manifest file, and the activator class. The create bundle activity takes the generated Java code, the service interface, the activator class, and the manifest file and compiles it all, creates the service JAR file and installs the new service into the OSGi framework. Figure 7.1 shows a summary of the average time taken by each activity for the experiments run under a regular networking environment and a WMN.

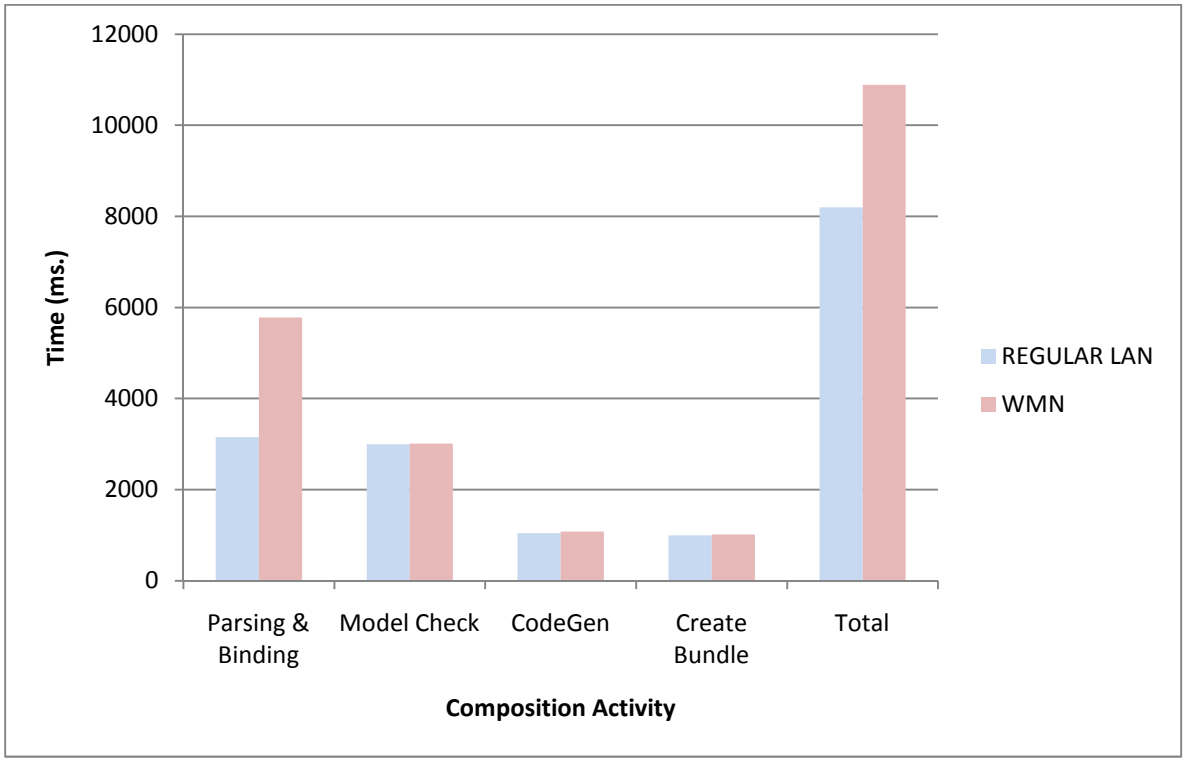

Figure 34 - Composition time overhead for MISS

For both cases, the figure show that the activity that consumes more time is parsing and binding. It also shows that there is significant difference in the parsing activity between the regular LAN network and the WMN. For the other activities, their time differences are almost similar. We believe a reason for this difference is the fact that the parsing activity includes network operations such as querying the UDDI server, fetching the services, and binding to them, which can take longer for networks with resource constraints like the WMN.

These results show evidence in support of *hypothesis 1*, that the regular network performs better than the WMN.

7.4 Total Parse Time

It is important to study the total time it takes to create a composite service as this operation imposes overhead. For that reason, a careful look at the parsing process of our composition framework is taken as this activity took the longest time to. The parsing process has the following activities: parsing, discovery, binding to the services, download the bundles JAR files, and generating WS stub code. The parsing activity reads data from the SSCL file to produce the executable Java code. The discovery queries the UDDI registry to get the binding information of the services. The binding activity gets the actual service and binds to it. The file download measures the average time to download the bundle JAR file for services provided in OSGi. The WS stub creation records the time to generate the stub code to communicate with a WS. The results of the parsing process under both networks are shown in Figure 7.2.

As shown in the figures, on average most activities consumes a relatively small amount of time with the exception of binding. The binding activity, use the partnerLinkTypes info in the SSCL file to get the services interface name, query the UDDI registry and use the services information to locate them and bind to them. As this activity does several networking operations, we believe this is the key factor for these differences. The fact that this activity executes faster under a regular LAN network than under a WMN, provides evidence in support of *Hypothesis 1*.

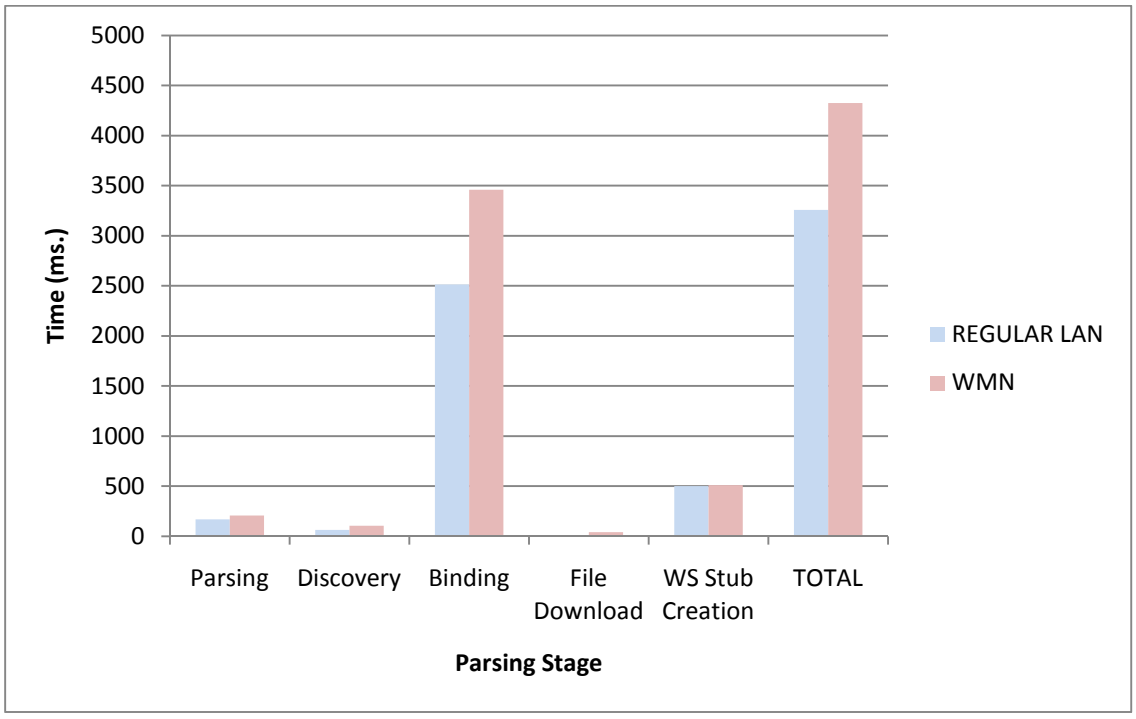

Figure 35 - Parse and binding overhead

7.5 Composite Services Performance Comparison

This subsection analyzes and compares the execution time of the three implementations of MISS using different SOA composition approaches and deployed under different network environments. The previous sections presented the overhead added by the composition framework. In this section, the execution times of the composite services using a single SOA are compared with the composite service generated automatically by our composition framework that relies on services of heterogeneous SOAs.

The execution times of MISS_OSGI, MISS_COMB, and MISS_WS are measured and their average computed. Figure 7.3 shows the average time and the standard deviation for the three composite services executed under a regular LAN network. Figure 7.4 shows data that includes the minimum and maximum. From the figure, it can be observed that MISS_OSGi has the best execution time performance, MISS_COMB had the second best performance while MISS_WS had the worst performance. The fact that WS are slower that OSGi, but are more popular and widely used, gives an upper bound on the acceptable execution time.

Having reduced the execution time from the worst-case scenario by introducing heterogeneous services, we believe is a great accomplishment. MISS_COMB also is closer to MISS_OSGi than to MISS_WS. These results provide evidence in support of *hypothesis 2*. The data for variability shows also that MISS_OSGI and MISS_COMB standard deviation and minimum/maximum ranges are close. It also shows MISS_COMB with less variability. Both MISS_OSGI and MISS_COMB show less variability than MISS_WS. These results provide evidence in support of *hypothesis 3.*

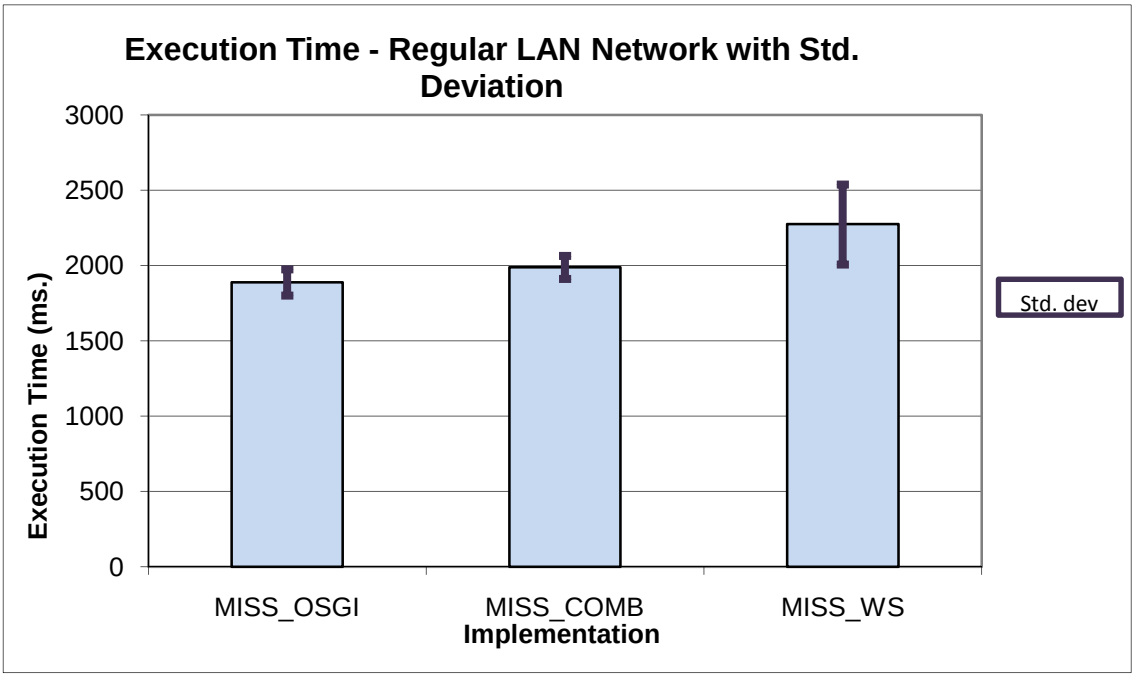

Figure 36 - Composite services in a regular network with their standard deviation.

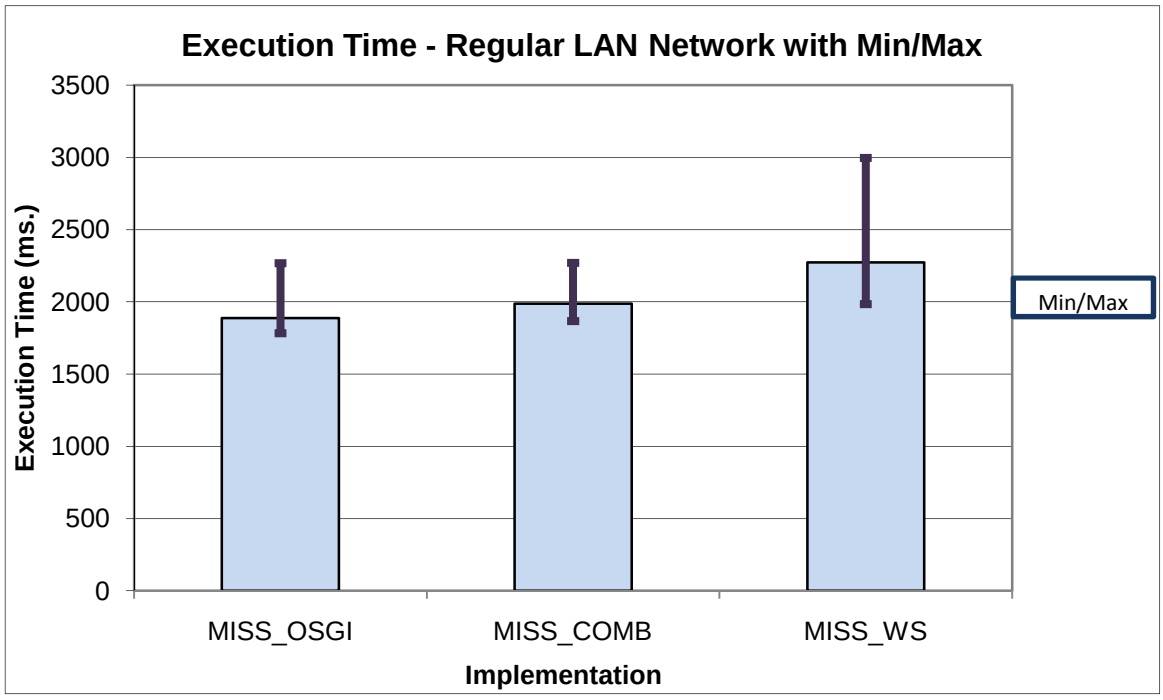

Figure 37 - Composite services in a regular network with their minimum and maximum.

Figure 7.5 shows the average time and the standard deviation for the three composite services running under a WMN with their standard deviation while figure 7.6 shows the data summary that includes the minimum and maximum. The time difference between MISS_OSGi and MISS_COMB is relatively small. The MISS_OSGi has the best performance of all three, closely followed by MISS_COMB, and MISS_WS had the worst performance. This data presents evidence in support of *hypothesis 2*. For WMN there is a dramatic difference in the execution time of MISS_WS, taking approximately 3 times longer than the other two versions of the composite service. In terms of variability, a similar result is obtained where MISS_COMB shows less variability than MISS_OSGi and MISS_WS shows the greatest variability. These results show evidence in support of *hypothesis 3*.

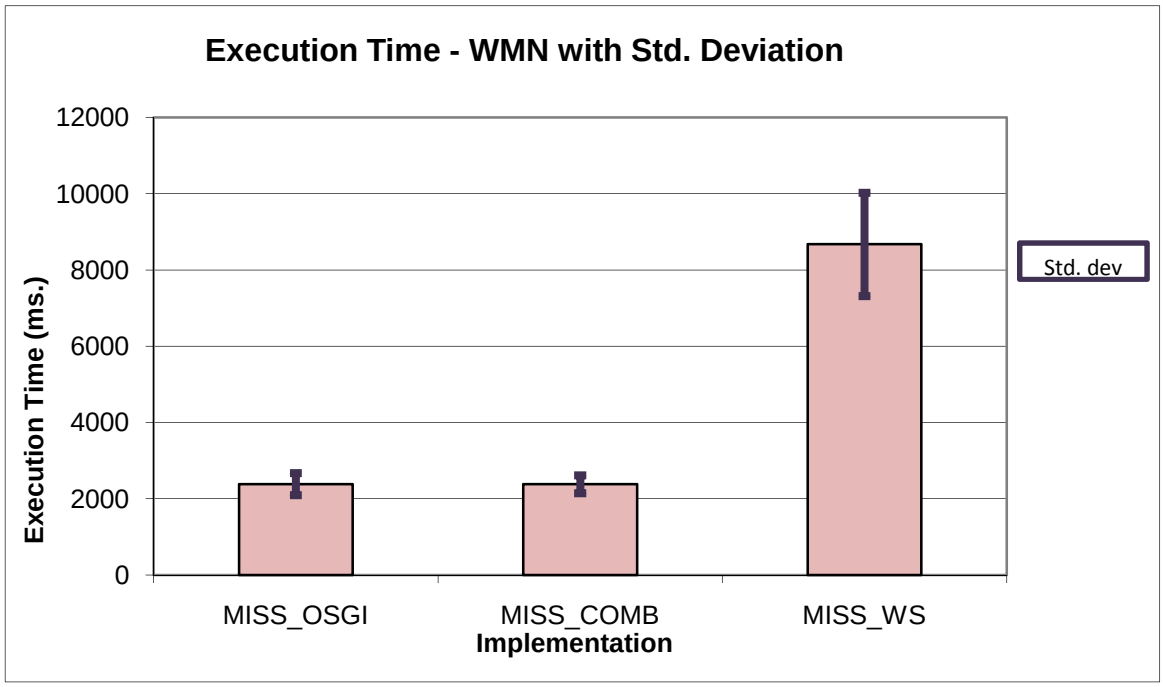

Figure 38 - Composite services in a WMN with their standard deviation.

Figure 39- Composite services in a WMN network with their minimum and maximum.

Figure 7.7 shows the result of the analysis of the performance of the MISS_OSGi composite service implementation in each type of network, with their respective standard deviations. Figure 7.8 show a summary with the minimum and maximum range values. The figures show that the MISS_OSGi composite service performs better under a regular network showing evidence that supports h*ypothesis 1*. In terms of variability, it is observed that the standard deviation is bigger for the WMN environment. The minimum, maximum range is also wider for the WMN, results that sustain *hypothesis 4* that the WMN shows more variability.

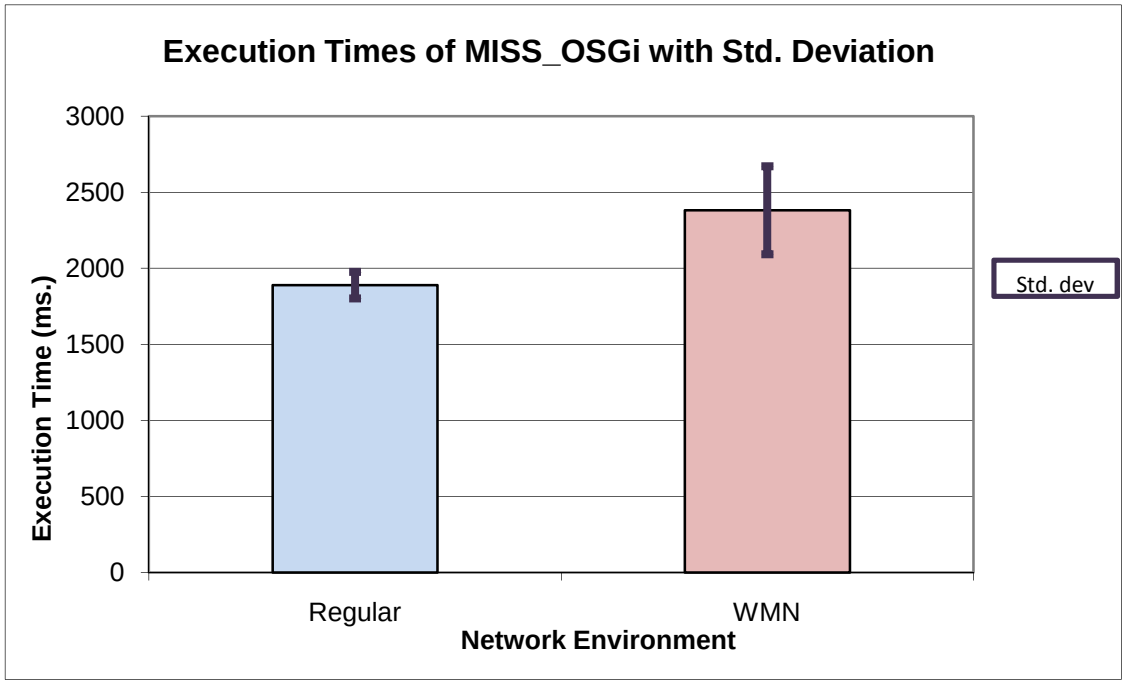

Figure 40 - MISS_OSGi in a regular network and a WMN with their std. deviation.

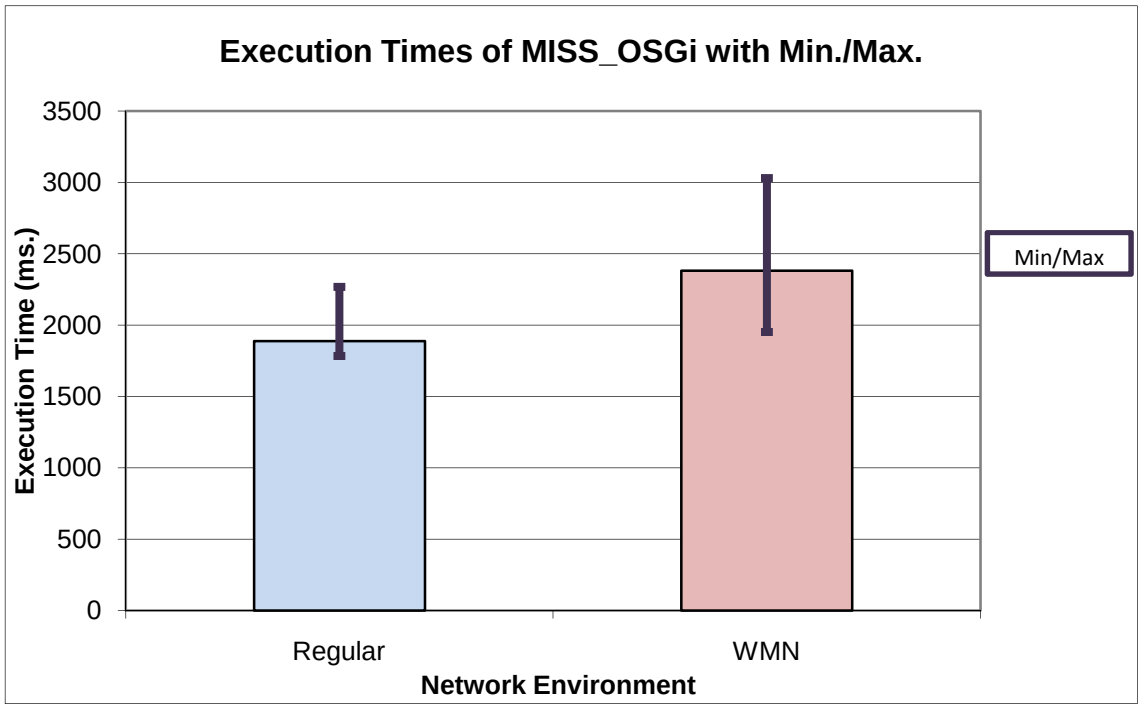

Figure 41 - MISS_OSGi in a regular network and a WMN with their minimum and maximum.

Figure 7.9 shows the performance analysis of the MISS_COMB composite service implementation under each type of network, with their respective standard deviations. Figure 7.10 shows a summary with the minimum and maximum values. Notice from the figures that the MISS_COMB composite service performs better under a regular network than under a WMN as evidence for h*ypothesis 1*. With respect to variability, the standard deviation is bigger for the WMNs as well as the minimum, maximum range, showing evidence in favor of *hypothesis 4*, that WMN have more variability.

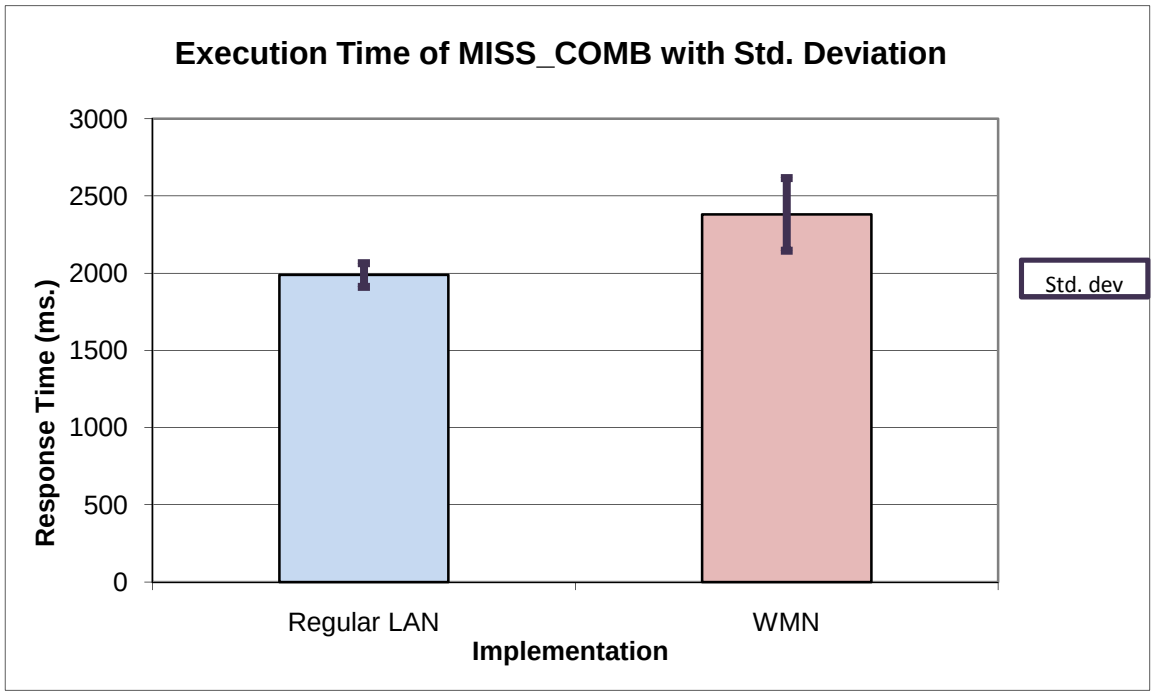

Figure 42 - MISS_COMB in a regular network and a WMN with their std. deviation

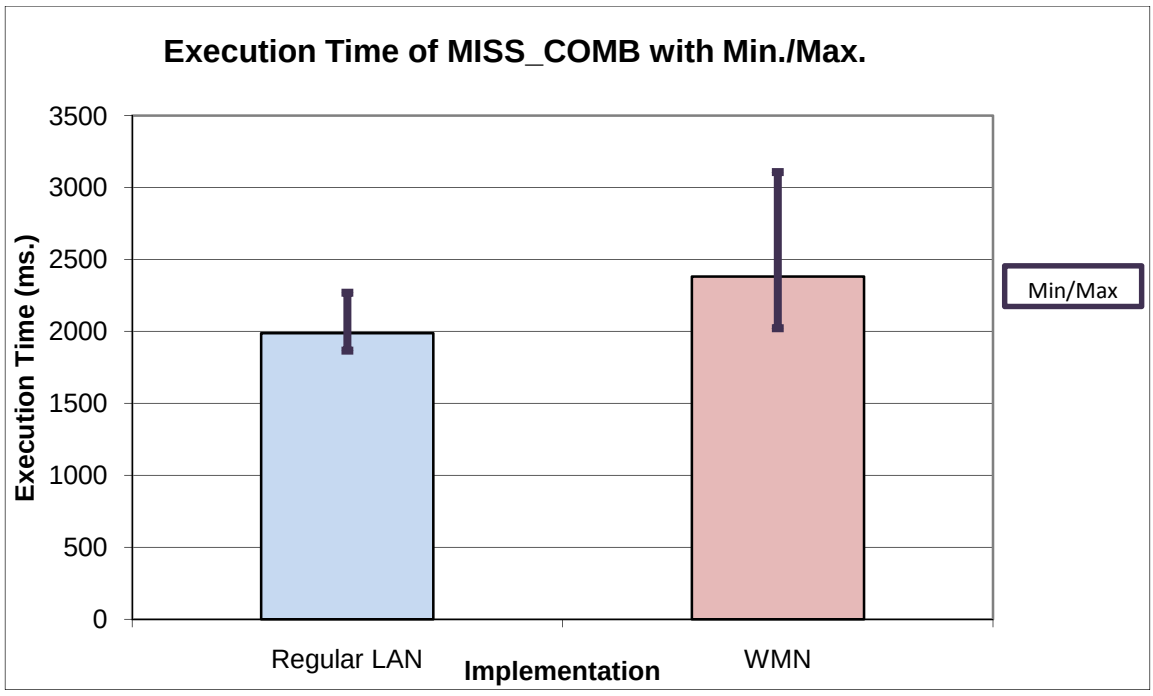

Figure 43 - MISS_COMB in a regular network and a WMN with their minimum and maximum

Figure 7.11 summarizes the performance results for the MISS_WS composite implementation in each type of network with their respective standard deviations. Figure 7.12 shows the minimum and maximum range. As in the previous two cases, it is observed from the figures that the MISS_WS composite service performs better under a regular network in support of h*ypothesis 1*. MISS_WS presents the case with the biggest difference in performance, having the service deployed in a WMN taking about three times as much time to execute compared to the same service deployed in a regular network. This evidence is the strongest in support of h*ypothesis 1*. With respect to variability, the standard deviation is bigger for the WMNs as well as the minimum, maximum range. Again, MISS_WS under a WMN provides the biggest difference of all cases, providing evidence that supports *hypothesis 4* that the WMN introduces more variability.

Figure 441 - MISS_WS in a regular network and a WMN with their standard deviation

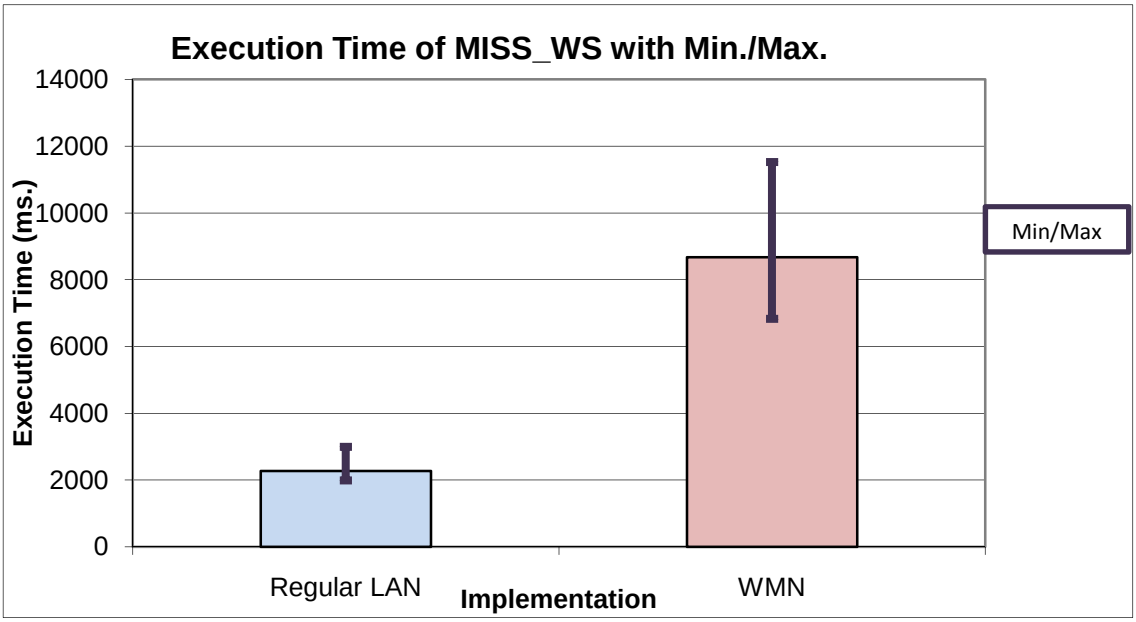

Figure 45 - MISS_WS in a regular network and a WMN with their minimum and maximum

Based on our hypothesis and the results obtained from our experiments we conclude that:

- The composition framework introduces overhead when creating a composite service, however this overhead is a onetime operation.
- The performance of the resulting MISS_COMB composite services automatically generated by the composition framework is closer to the best-case scenario of MISS_OSGi than to the worst-case scenario of MISS_WS.
- *Hypothesis 1* holds as services deployed in the regular network consistently performed better than those deployed under the WMN in our experiments.
- *Hypothesis 2* holds as MISS_OSGi performed better than MISS_COMB and both performed better than MISS_WS in both kinds of network environment in our experiments.

- *Hypothesis 3* holds as the implementation with less variability under both types of network is MISS_COMB in our experiments
- *Hypothesis 4* holds as the WMN introduced more variability that a regular network in our experiments.

CHAPTER 8. CONCLUSIONS AND FUTURE WORK

This chapter summarizes our work, contributions, and present future work.

8.1 Summary

Pervasive environments such as the Smart Home rely on the SOC paradigm for the development of services and applications. Different SOA standards have been proposed and developed following the SOC principles of language-independence and platform-independence. However, interactions among services of heterogeneous SOAs are a non-trivial task. As the number of services and the complexity of applications increases, combining them becomes more necessary. The goal of this work is to provide a framework that automatically creates composite services supporting heterogeneous SOAs, uses model checking techniques to ensure the safety of composite services and their interactions, and has acceptable performance. Pervasive computing environments such as the smart home services would benefit of this type of solution as it can reduce the development time, cost, and learning curve.

We present a set of fundamental services such as a notification, medicine management, and a middleware service as an example. The Medicine Information Support System (MISS) is used as our main composite service example in the discussions throughout this dissertation. The details of our composition framework for services of heterogeneous SOAs are provided. At a high level, our framework operates as follows: a description of the composite service is provided using the simple XML-based language SSCL, the framework searches for services within the context or look for them in a customized service registry, gets the binding information, checks the safety of the composite service, and automatically generates the implementation. A set of safety criteria is defined and the composition framework checks that composite services and their interactions comply with it. For checking compliance, formal software analysis techniques of model checking are integrated into our composition framework. Based on their characteristics services are divided into baseline and extended. A semi-automatic approach for checking baseline services is presented along with an automatic approach for checking extended services. A performance analysis is provided which

compares the execution times of composite services using different composition strategies. The examples used are based on MISS and details on the different composites using current composition techniques are compared with the composite services generated automatically by our composition framework. A comparison of the execution performance of these services is provided. The experiment measured the overhead that the composition process adds. These experiments were conducted under two networking environments: a regular LAN network and a wireless mesh network. The two networking environments are used to contrast how the composition framework would perform under regular network condition and under a constrained networking environment.

8.2 Contributions

The key contributions of this work are:

- A set of requirements and capabilities for scripting languages used for orchestrations with a justification of why we understand these should be part of any SOA standard.

- The design, implementation, and testing of a set of essential services for pervasive environments such as the smart home.

- A language for specifying the workflow of composite services called the Simple Service Composition Language (SSCL). The SSCL language provides is platform, language, and SOA independence. It is inspired in existing technologies, complies with the requirements we established for scripting languages for orchestrations providing also SOA-independence, allowing for compositions of services of heterogeneous SOAs.

- A customized universal service registry where services information such the description and binding information can be stored, searched, and retrieved. This registry allows registering services of heterogeneous SOAs and is essential for our framework to work automatically.

- The SSCL2OSGi composition framework, that accepts SSCL files as input and automatically produces an OSGi service that implements the composite using services of heterogeneous SOAs. The framework automatically searches the services, performs a safety check, creates the composite service using services from heterogeneous SOAs, installs the composite service, and deploys it.

- A combined model-checking approach for checking the safety of baseline and extended services. Baseline services are checked for compliance with standard and custom safety criteria in a semi-automated way. Extended services are checked for standard safety criteria in a fully automated way. The automatic check for extended services is integrated as part of the composition framework. Only services that pass all the safety checks are implemented.

- SSCL is a language inspired in WS-BPEL and SCDL. We claim and justify that the semantic of SSCL and WS-BPEL are compatible and that the design of SSCL allows translating SSCL files into oWFN. Compatibility among SSCL files and WS-BPEL files is defined. We claim and show that the oWFN produced from a WS-BPEL file and a compatible SSCL file are the same. With these results, it is shown that the composition framework can be used with SSCL, produce an oWFN model of composite services, and use existing tools to analyze oWFN.

- A performance study of the composition framework execution time and the overhead it adds. Several examples are provided by creating different composite services. The execution times of these different composite services are compared. These composites are deployed under a regular network and under a WMN constrained environment. A detailed analysis and a summary of the performance results are provided.

8.3 Future Work

- Extend the composition framework to support other SOAs and test the performance of these additions

- Extend SSCL to support other commands without compromising its simplicity and SOA independence.

- Extend the formal analysis to have automated modeling of baseline services

REFERENCES

[1] N. Noury, G. Virone, P. Barralon, J. Ye, V. Rialle, and J. Demongeot, “New trends in health smart homes,” 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry, 2003, pp. 118-127.

[2] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-Oriented Computing: State of the Art and Research Challenges,” *Computer*, vol. 40, 2007, pp. 38-45.

[3] S. Dustdar and W. Schreiner, “A survey on web services composition,” *International Journal on Web and Grid Services*, vol. 1, 2005, pp. 1-30.

[4] D. Marples and P. Kriens, “The Open Services Gateway Initiative: an introductory overview,” *IEEE Communications Magazine*, vol. 39, 2001, pp. 110-114.

[5] “OSOA - Open SOA Collaboration.” Available: http://www.osoa.org/display/Main/Home. [Accessed: Jun. 23, 2010].

[6] “OASIS - Who We Are - Mission.” Available: http://www.oasis-open.org/who/. [Accessed: Jun. 23, 2010].

[7] “IEEE SOA Standards.” Available: http://www.soa-standards.org/. [Accessed: Jun. 23, 2010].

[8] “OSGi Alliance | Main / OSGi Alliance.” Available: http://www.osgi.org/Main/HomePage. [Accessed: Jun. 23, 2010].

[9] “OASIS SOA Reference Model TC,” Jun. 2010. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm. [Accessed: Jun. 23, 2010].

[10] N. Ibrahim and F.L. Mouel, " A Survey on Service Composition Middleware in Pervasive Environments", International Journal of Computer Science Issues, Volume 1, August 2009, pp. 1-12.

[11] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, “Automatic Web Services Composition Using SHOP2,” Twelfth World Wide Web Conference, 2003.

[12] “Service Component Architecture Home - Open SOA Collaboration.” Available: http://www.osoa.org/display/Main/Service+Component+Architecture+Home. [Accessed: Jun. 23, 2010].

[13] M.B. Dwyer, J. Hatcliff, R. Robby, C.S. Pasareanu, and W. Visser, “Formal Software Analysis Emerging Trends in Software Model Checking,” Future of Software Engineering, 2007. FOSE '07, 2007, pp. 120-136.

[14] T. Koskela and K. Väänänen-Vainio-Mattila, “Evolution towards smart home environments: empirical evaluation of three user interfaces,” Personal and Ubiquitous Computing, vol. 8, Jul. 2004, pp. 234-240.

[15] S. Dengler, A. Awad, and F. Dressler, “Sensor/Actuator Networks in Smart Homes for Supporting Elderly and Handicapped People,” Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 02, IEEE Computer Society, 2007, pp. 863-868.

[16] M. Satyanarayanan, “Pervasive Computing: Vision and Challenges,” IEEE Personal Communications, vol. 8, 2001, pp. 10--17.

[17] D. Saha and A. Mukherjee, “Pervasive Computing: A Paradigm for the 21st Century,” Computer, vol. 36, 2003, pp. 25-31.

[18] S. Helal, “Programming Pervasive Spaces,” IEEE Pervasive Computing, vol. 4, 2005, pp. 84-87.

[19] J. King, R. Bose, H. Yang, S. Pickles, and A. Helal, “Atlas: A Service-Oriented Sensor Platform Hardware and Middleware to Enable Programmable Pervasive Spaces.” Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006.

[20] B.A. Miller, T. Nixon, C. Tai, and M.D. Wood, “Home networking with Universal Plug and Play,” Communications Magazine, IEEE, vol. 39, Dec. 2001, pp. 104 -109.

[21] R. Gupta, S. Talwar, and D.P. Agrawal, “Jini Home Networking: A Step toward Pervasive Computing,” Computer, vol. 35, 2002, pp. 34-40.

[22] C. Lee, D. Nordstedt, and S. Helal, “Enabling smart spaces with OSGi,” Pervasive Computing, IEEE, vol. 2, 2003, pp. 89-94.

[23] H. Elzabadani, A. Helal, B. Abdulrazak, and E. Jansen, “Self-sensing spaces: smart plugs for smart environments,” Proceedings of the 3rd International Conference on Smart Homes and Health Telematics, 2005.

[24] “Iowa State University Smart Home Project.” Available: http://smarthome.cs.iastate.edu. [Accessed: Jun. 23, 2010].

[25] S. Helal, “The Gator Tech Smart House: A Programmable Pervasive Space,” Computer, Vol. 38, No. 3. (2005), pp. 50-60.

[26] R. Bose, J. King, S. Pickles, H. Elzabadani, and A. Helal, “Building Plug-and-Play Smart Homes Using the Atlas Platform.” In 4th International Conference on Smart Homes and Health Telematic, 2006.

[27] C.D. Kidd, R. Orr, G.D. Abowd, C.G. Atkeson, Irfan A, B. Macintyre, E. Mynatt, T.E. Starner, W. Newstetter, “The Aware Home: A Living Laboratory for Ubiquitous Computing Research,” Cooperative Buildings: Integrating Information, Organizations and Architecture, 1999.

[28] H. Kautz, L. Arnstein, G. Borriello, O. Etzioni, and D. Fox, “An overview of the assisted cognition project,” Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder, 2002.

[29] M. Papazoglou and J.-J. Dubray. A survey of web service technologies. Technical Report DIT-04-058, Department of Information and Communication Technology, University of Trento, June 2004.

[30] “seekda! - About seekda.” Available: http://seekda.com/. [Accessed: Jun. 23, 2010].

[31] “X Methods.” Available: http://www.xmethods.net/ve2/index.po. [Accessed: Jun. 23, 2010].

[32] “List of Web service specifications.” Available: http://www.worldlingo.com/ma/enwiki/en/List_of_Web_service_specifications. [Accessed: Jun. 23, 2010].

[33] “Web Tools Platform (WTP) Project.” Available: http://www.eclipse.org/webtools/. [Accessed: Jun. 23, 2010].

[34] S.M. Kim and M. Rosu, “A Survey of Public Web Services,” E-Commerce and Web Technologies, 2004, pp. 96-105.

[35] “Apache jUDDI - Welcome to jUDDI.” Available: http://ws.apache.org/juddi/. [Accessed: Jun. 23, 2010].

[36] “OpenUDDI » Welcome.” Available: http://openuddi.sourceforge.net/. [Accessed: Jun. 23, 2010].

[37] “UDDI4J - Links.” Available: http://uddi4j.sourceforge.net/. [Accessed: Jun. 23, 2010].

[38] E. Topalis, S. Koubias, G. Papadopoulos, and I. Nikiforakis, “A Novel Architecture for Remote Home Automation e-Services on an OSGi Platform via High-Speed Internet Connection Ensuring QoS Support by Using RSVP,” IEEE Transactions on Consumer Electronics, vol. 48, pp. 825--833.

[39] OSGi Alliance, “OSGi Service Platform Release 4,” 2007. Available: http://www.osgi.org/Release4/HomePage. [Accessed: Jun. 23, 2010].

[40] “Knopflerfish OSGi - open source OSGi service platform.” Available: http://www.knopflerfish.org/. [Accessed: Jun. 23, 2010].

[41] “Oscar Bundle Repository.” Available: http://oscar-osgi.sourceforge.net/. [Accessed: Jun. 23, 2010].

[42] “Eclipse Equinox.” Available: http://www.eclipse.org/equinox/. [Accessed: Jun. 23, 2010].

[43] “Apache Felix - Index.” Available: http://felix.apache.org/site/index.html. [Accessed: Jun. 23, 2010].

[44] T. Gu, H. Pung, and D. Zhang, “Toward an OSGi-based infrastructure for context-aware applications,” IEEE Pervasive Computing, vol. 3, 2004, pp. 66-74.

[45] E. Karakoc, K. Kardas, and P. Senkul, “A Workflow-Based Web Service Composition System,” International Conference on Web Intelligence and Intelligent Agent Technology, Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 113-116.

[46] J. Rao and X. Su, “A Survey of Automated Web Service Composition Methods,” Semantic Web Services and Web Process Composition, 2005, pp. 43-54.

[47] S. Ponnekanti and A. Fox, “SWORD: A developer toolkit for web service composition,” Proceedings of the 11th International WWW Conference (WWW2002), 2002.

[48] A. Wood, J. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, and R. Stoleru, “Context-aware wireless sensor networks for assisted living and residential monitoring,” IEEE Network, vol. 22, 2008, pp. 26-33.

[49] R. Redondo, A. Vilas, M. Cabrer, J. Arias, and M. Lopez, “Enhancing Residential Gateways: OSGi Services Composition,” International Conference on Consumer Electronics, 2007, pp. 1-2.

[50] J. Anke and C. Sell, “Seamless Integration of Distributed OSGi Bundles into Enterprise Processes using BPEL,” Kommunikation in Verteilten Systemen, 2007.

[51] C. Lee, S. Ko, E. Kim, and W. Lee, “Enriching OSGi Service Composition with Web Services,” IEICE Transactions, vol. 92-D, 2009, pp. 1177-1180.

[52] “Apache Tuscany.” Available: http://tuscany.apache.org/. [Accessed: Jun. 23, 2010].

[53] “Fabric3 - Open Source SCA.” Available: http://www.fabric3.org/. [Accessed: Jun. 23, 2010].

[54] “Newton Framework.” Available: http://newton.codecauldron.org/. [Accessed: Jun. 23, 2010].

[55] P. Cousot and R. Cousot, “Refining Model Checking by Abstract Interpretation,” Automated Software Engineering, vol. 6, Jan. 1999, pp. 69-95.

[56] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. Sipma, and T. Uribe, “STeP: Deductive-algorithmic verification of reactive and real-time systems,” Computer Aided Verification, 1996, pp. 415-418.

[57] C. Flanagan and S. Qadeer, “Predicate abstraction for software verification,” Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2002, pp. 191-202.

[58] S. Khurshid, C. PĂsĂreanu, and W. Visser, “Generalized Symbolic Execution for Model Checking and Testing,” Tools and Algorithms for the Construction and Analysis of Systems, 2003, pp. 553-568.

[59] C. Flanagan and P. Godefroid, “Dynamic partial-order reduction for model checking software,” SIGPLAN, vol. 40, 2005, pp. 110-121.

[60] I. Akyildiz and X. Wang, “A survey on wireless mesh networks,” IEEE Communications Magazine, vol. 43, 2005, pp. S23-S30.

[61] N. Lohmann, “A Feature-Complete Petri Net Semantics for WS-BPEL 2.0,” Web Services and Formal Methods, 2008, pp. 77-91.

[62] P. Massuthe and D. Weinberg, “Fiona: A Tool to Analyze Interacting Open Nets,” Proceedings of the 15th German Workshop on Algorithms and Tools for Petri Nets, 2008, pp. 99-104.

[63] M.J. May, C.A. Gunter, and I. Lee, “Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies,” Proceedings of the 19th IEEE workshop on Computer Security Foundations, 2006, pp. 85-97.

[64] M. Lee, J. Zheng, Y. Ko, and D. Shrestha, “Emerging standards for wireless mesh technology,” IEEE Wireless Communications, vol. 13, 2006, pp. 56- 63.

[65] I.F. Akyildiz, X. Wang, and W. Wang, “Wireless mesh networks: a survey,” Computer Networks, vol. 47, Mar. 2005, pp. 445-487.

[66] K. Havelund and T. Pressburger, “Model checking JAVA programs using JAVA PathFinder,” International Journal on Software Tools for Technology Transfer (STTT), vol. 2, Mar. 2000, pp. 366-381.

[67] “Spin - Formal Verification.” Available: http://spinroot.com/. [Accessed: Jun. 23, 2010].

[68] J.M. Reyes Álamo, J. Wong, R. Babbitt, and C. Chang, “MISS: Medicine Information Support System in the Smart Home Environment,” Smart Homes and Health Telematics, 2008, pp. 185-199.

[69] J.M. Reyes Álamo, J. Wong, R. Babbitt, H. Yang, and C. Chang, “Using Web Services for Medication Management in a Smart Home Environment,” Ambient Assistive Health and Wellness Management in the Heart of the City, 2009, pp. 265-268.

[70] I. Korhonen, J. Parkka, and M. Van Gils, “Health monitoring in the home of the future,” IEEE Engineering in Medicine and Biology Magazine, vol. 22, 2003, pp. 66-73.

[71] “The Wireless Hive Networks Manifesto.” Available: http://www.ieee-whnc.org/WHN-Manifesto.pdf. [Accessed: Jun. 23, 2010].

[72] “Phidgets Inc. - Unique and Easy to Use USB Interfaces.” Available: http://www.phidgets.com/. [Accessed: Jun. 23, 2010].

[73] M. Vastenburg, D. Keyson, and H. de Ridder, “Considerate home notification systems: a field study of acceptability of notifications in the home,” Personal and Ubiquitous Computing, vol. 12, Nov. 2008, pp. 555-566.

[74] “Medical Guardian Medical Alert Systems.” Available: http://www.medicalguardian.com/. [Accessed: Jun. 23, 2010].

[75] “Home Security Store | Home Security Camera | Wireless Security Systems.” Available: http://www.homesecuritystore.com/. [Accessed: Jun. 23, 2010].

[76] “AlarmCare | Senior Medical Alarm • Emergency Alert Button • Personal Alarm System.” Available: http://www.myalarmcare.com/. [Accessed: Jun. 23, 2010].

[77] “Building smart services for smart home.”. In Proceedings of 4th IEEE International Workshop on Networked Appliances, 2002, pp. 215–224.

[78] Tanmoy Sarkar, “A Web–based Architecture for Usability of Service Oriented Environments,” 6th International Conference on Smart Homes and Health Telematics, 2008.

[79] J.M. Reyes Álamo and J. Wong, “Service-Oriented Middleware for Smart Home Applications,” Wireless Hive Networks, Austin, Texas: 2008.

[80] C.D. Nugent, D. Finlay, R. Davies, C. Paggetti, E. Tamburini, and N. Black, “Can Technology Improve Compliance to Medication?” 3rd International Conference on Smart Homes and Health Telematics, 2005.

[81] M. Aiello, “The Role ofWeb Services at Home,” International Conference on Internet and Web Applications and Services, 2006, p. 164.

[82] A. Bottaro, E. Simon, S. Seyvoz, and A. Gerodolle, “Dynamic Web Services on a Home Service Platform,” 22nd International Conference on Advanced Information Networking and Applications, 2008, pp. 378-385.

[83] K. Romer and F. Mattern, “The design space of wireless sensor networks,” IEEE Wireless Communications, vol. 11, 2004, pp. 54-61.

[84] G.J. Holzmann, The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley Professional, 2003.

[85] R.M. Babbitt, “A service-oriented privacy model for smart home environments,” Iowa State University, 2006.

[86] OCR, “Summary of the HIPAA Privacy Rule.” Available: http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/privacysummary.pdf. [Accessed: Jun. 23, 2010].

[87] D. Wan, “Magic Medicine Cabinet: A Situated Portal for Consumer Healthcare,” First International Symposium on Handheld and Ubiquitous Computing, vol. 1707, 1999, p. 352-355

[88] J. Brusey, M. Harrison, C. Floerkemeier, and M. Fletcher, “Reasoning about uncertainty in location identification with RFID,” Workshop on Reasoning with Uncertainty in Robotics at IJCAI, 2003.

[89] C. Floerkemeier, M. Lampe, and T. Schoch, “The Smart Box Concept for Ubiquitous Computing Environments,” in Smart Objects Conference, 2003.

[90] F. Siegemund and C. Floerkemeier, “Interaction in Pervasive Computing Settings using Bluetooth-enabled Active Tags and Passive RFID Technology together with Mobile Phones”, IEEE PerCom, 2003.

[91] C. Paggetti and E. Tamburini, “Remote Management of Integrated Home Care Services: the DGHome Platform,” 3rd International Conference on Smart Homes and Health Telematics, 2005.

[92] “e-pill Medication Reminders: Pill Dispenser, Vibrating Watch, Pill Box Timer & Alarms.” Available: http://www.epill.com/. [Accessed: Jun. 23, 2010].

[93] M. Lampe and C. Flörkemeier, “The Smart Box application model,” Advances in Pervasive Computing, 2004, pp. 351-356.

[94] A.Y. Szeto and J.A. Giles, “Improving oral medication compliance with an electronic aid,” IEEE Engineering in Medicine and Biology Magazine: The Quarterly Magazine of the Engineering in Medicine & Biology Society, vol. 16, Jun. 1997, pp. 48-54, 58.

[95] D. Vergnes, S. Giroux, and D. Chamberland-Tremblay, “Interactive Assistant for Activities of Daily Living,” 3rd International Conference on Smart Homes and Health Telematics, 2005.

[96] V. Fook, J. Tee, K. Yap, A. Phyo Wai, J. Maniyeri, B. Jit, and P. Lee, “Smart Mote-Based Medical System for Monitoring and Handling Medication Among Persons with Dementia,” Pervasive Computing for Quality of Life Enhancement, 2007, pp. 54-62.

[97] “WebMD Mobile for Apple iPhone.” Available: http://www.webmd.com/mobile. [Accessed: Jun. 23, 2010].

[98] “Google Health.” Available: http://www.google.com/health/. [Accessed: Jun. 23, 2010].

[99] “HealthVault : Home.” Available: http://www.healthvault.com/. [Accessed: Jun. 23, 2010].

[100] L.M. Ni, Y. Liu, Y.C. Lau, and A.P. Patil, “LANDMARC: Indoor Location Sensing Using Active RFID,” Wireless Networks, vol. 10, Nov. 2004, pp. 701-710.

[101] “U S Food and Drug Administration Home Page.” Available: http://www.fda.gov/. [Accessed: Jun. 23, 2010].

[102] “Welcome to PDR.NET - The Physicians' Desk Reference web site providing prescription drug information and more.” Available: http://www.pdr.net/. [Accessed: Jun. 23, 2010].

[103] R. Want, “Enabling Ubiquitous Sensing with RFID,” Computer, vol. 37, 2004, pp. 84-86.

[104] A. Sarriff, N.A. Aziz, Y. Hassan, P. Ibrahim, and Y. Darwis, “A study of patients' self-interpretation of prescription instructions,” Journal of Clinical Pharmacy and Therapeutics, vol. 17, Apr. 1992, pp. 125-128.

[105] J.M. Mazzullo, L. Lasagna, and P.F. Griner, “Variations in interpretation of prescription instructions. The need for improved prescribing habits,” JAMA: The Journal of the American Medical Association, vol. 227, Feb. 1974, pp. 929-931.

[106] “XML Schema Part 2: Datatypes Second Edition.” Available: http://www.w3.org/TR/xmlschema-2/. [Accessed: Jun. 23, 2010].

[107] O. Gruber, B.J. Hargrave, J. McAffer, P. Rapicault, and T. Watson, “The Eclipse 3.0 platform: adopting OSGi technology,” IBM Systems Journal Archive, vol. 44, 2005, pp. 289-299.

[108] “Apache Axis - Web Services.” Available: http://ws.apache.org/axis/. [Accessed: Jun. 23, 2010].

[109] A. Ng, S. Chen, and P. Greenfield, “An Evaluation of Contemporary Commercial SOAP Implementations,” In AWSA, 2004, pp. 64--71.

[110] “Invoking Web services with Java clients,” Nov. 2003. Available: http://www.ibm.com/developerworks/webservices/library/ws-javaclient/index.html. [Accessed: Jun. 23, 2010].

[111] J.M. Reyes Álamo, H. Yang, J. Wong, R. Babbitt, and C.K. Chang, “Support for Medication Safety and Compliance in Smart Home Environments,” International Journal of Advanced Pervasive and Ubiquitous Computing, vol. 1, 2009, pp. 42-60.

[112] H. Schlingloff, A. Martens, and K. Schmidt, “Modeling and Model Checking Web Services,” Electronic Notes in Theoretical Computer Science, vol. 126, Mar. 2005, pp. 3-26.

[113] X. Fu, T. Bultan, and J. Su, “Analysis of interacting BPEL web services,” Proceedings of the 13th international conference on World Wide Web, New York, NY, USA: ACM, 2004, pp. 621-630.

[114] C. Gao, R. Liu, Y. Song, and H. Chen, “A Model Checking Tool Embedded into Services Composition Environment,” Grid and Cooperative Computing, International Conference on, Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 355-362.

[115] J.M. Reyes Álamo, H. Yang, J. Wong, and C.K. Chang, “Automatic Service Composition with Heterogeneous Service-Oriented Architectures,” ICOST 2010.

[116] R. Hamadi and B. Benatallah, “A Petri net-based model for web service composition,” Proceedings of the 14th Australasian database conference - Volume 17, Adelaide, Australia: Australian Computer Society, Inc., 2003, pp. 191-200.

[117] K. Schmidt, “Controllability of open workflow nets,” IN: EMISA. LNI, BONNER KÖLLEN VERLAG, vol. 75, 2005, pp. 236--249.

[118] G.J. Holzmann, “The Model Checker SPIN,” Software Engineering, vol. 23, 1997, pp. 279-295.

[119] T. Murata, “Petri nets: Properties, analysis and applications,” Proceedings of the IEEE, vol. 77, 2002, pp. 580, 541.

[120] N. Lohmann, “A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN,” IN WS-FM 2007, LNCS, 2007.

[121] P. Massuthe, W. Reisig, and K. Schmidt, “An Operating Guideline Approach to the SOA,” Annals of Mathematics, Computing & Teleinformatics, vol. 1, 2005, pp. 35--43.

[122] C. Stahl, A Petri net semantics for BPEL, Technical Report 188, Humboldt-Universität zu Berlin, June 2005.

[123] S. Hinz, K. Schmidt, and C. Stahl, “Transforming BPEL to Petri Nets,” Business Process Management, 2005, pp. 220-235.

[124] “BPEL2oWFN - GNU Project - Free Software Foundation.” Available: http://www.gnu.org/software/bpel2owfn/. [Accessed: Jun. 23, 2010].

[125] “Fiona Model Checker.” Available: http://www.service-technology.org/fiona. [Accessed: Jun. 23, 2010].

[126] “Self Organizing Wireless Mesh Networks - Microsoft Research.” Available: http://research.microsoft.com/en-us/projects/mesh/. [Accessed: Jun. 23, 2010].

APPENDIX A. Formal Syntax of SSCL

Syntax of SSCL

Notation:
O(x) ::= empty | x
#(x) ::= any number of x
P(x) ::= x #(x)
U(x,y) ::= any order of x and y

SSCL *::= U(O(PartnerLinkTypes), O(PartnerLinks), O(Variables), O(Activities))*

PartnerLinkTypes *::= "<parnerLinkTypes>" P(PartnerLinkType) "</parnerLinkTypes>"*

PartnerLinkType *::= "<partnetLinkType name =" serviceInterfaceName "/>"*

PartnerLinks *::= "<partnerLinks>" P(PartnerLink) "</partnerLinks>"*

PartnerLink *::= "<partnerLink name =" serviceName "partnerLinkType =" serviceInterfaceName "/>"*

Variables *::= "<variables>" P(Variable) "</variables>"*

Variable*::= "<variable name =" variableName "type ="xsd:TYPE "/>"*

Activities *::= P(basic-activities | structured-activities)*

Basic-activities *::= Invoke | Assign*

Structured-activities *::= Sequence | If | While | RepeatUntil |ForEach*

Invoke *::= "<invoke" invoke-attributes ">" "</invoke>"*

Invoke-attributes *::= U(partnerLink, operation, O(inputVariable), O(outputVariable))*

Assign *::= "<assign>" P(copy) "</assign>"*

Copy *::= "<copy>" From To "</copy>" (the definition of "from" and "to" are skipped here)*

From *::= "<from>" copyConstant | copyVariable "</from>"*

To *::= “<to variable =” copyToVariable ”/>”*

Sequence *::= “<sequence>" Activities “</sequence>"*

If *::= “<if >" Condition Activities #(elseif) O(else) “</if>"*

Condition *::= “<condition>” Boolean | not operand | Comparison “</condition>”*

Boolean *::= true | false*

Comparison *::= leftOperand conditionSymbol rightOperand*

ConditionSymbol *::= < | <= | == | >= | > | or | and*

While *::= “<while>" Condition Activities“</while>"*

RepeatUntil *::= “<repeatUntil>" Activities Condition “</repeatUntil >"*

forEach *::= “<forEach counterName =” counterName “>" U(StartCounterValue, FinalCounterValue, Scope “</forEach>"*

StartCounterValue *::= “<startCounterValue>” value “</startCounterValue>”*

FinalCounterValue *::= “<finalCounterValue>” value “</finalCounterValue>”*

Scope *::= “<scope>" scope-elements “</scope>"*

Scope-elements *::= U(O(partnerLink),O(variables), Activities)* □

APPENDIX B. MISS Workflow in SSCL

```
<!-- SSCL file of the MISS system-->
    <process name="MISSSSCL">
    <!-- ~~~~~~~     PARTNER LINK TYPE DEFINITION ~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <partnerLinkTypes>
        <partnerLinkType name="MISSSSCL"/>
        <partnerLinkType name="PharmacyService"/>
        <partnerLinkType name="SpeechService"/>
        <partnerLinkType name="MCDService"/>
        <partnerLinkType name="RFIDService"/>
        <partnerLinkType name="MedicineSHService"/>
        <partnerLinkType name="NotificationService"/>
    </partnerLinkTypes>
    <!-- PARTNERLINKS DEFINITIONS -->
    <partnerLinks>
        <partnerLink name="client" partnerLinkType="MISSSSCL"/>
        <partnerLink name="pharmacyService" partnerLinkType="PharmacyService"/>
        <partnerLink name="speechService" partnerLinkType="SpeechService"/>
        <partnerLink name="mcdService" partnerLinkType="MCDService"/>
        <partnerLink name="rfidService" partnerLinkType="RFIDService"/>
        <partnerLink name="medSHService" partnerLinkType="MedicineSHService"/>
        <partnerLink name="notificationService" partnerLinkType="NotificationService"/>
    </partnerLinks>
    <!-- VARIABLES  ================================================== -->
    <variables>
        <variable name="input" type="xsd:string"/>
        <variable name="output" type="xsd:string"/>
        <variable name="rfidTag" type="xsd:string"/>
```

```
        <variable name="pharmacyTag" type="xsd:string"/>
        <variable name="firstName" type="xsd:string"/>
        <variable name="lastName" type="xsd:string"/>
        <variable name="newPrescription" type="xsd:string"/>
        <variable name="quantity" type="xsd:string"/>
        <variable name="frequency" type="xsd:string"/>
        <variable name="duration" type="xsd:string"/>
        <variable name="options" type="xsd:string"/>
        <variable name="comments" type="xsd:string"/>
        <variable name="oldPrescription" type="xsd:string"/>
        <variable name="food" type="xsd:string"/>
        <variable name="condition" type="xsd:string"/>
        <variable name="hasConflicts" type="xsd:boolean"/>
        <variable name="speakMessage" type="xsd:string"/>
        <variable name="conflictString" type="xsd:string"/>
        <variable name="noConflictString" type="xsd:string"/>
        <variable name="email" type="xsd:string"/>
    </variables>
    <!-- ORCHESTRATION LOGIC ============================================== -->
    <sequence name="main">
        <invoke partnerLink="rfidService" operation="getRFIDTag" outputVariable="rfidTag"/>
        <invoke partnerLink="pharmacyService" operation="setTag" inputVariable="rfidTag"/>
        <invoke partnerLink="pharmacyService" operation="getFirstName" outputVariable="firstName"/>
        <invoke partnerLink="pharmacyService" operation="getLastName" outputVariable="lastName"/>
        <invoke partnerLink="pharmacyService" operation="getMedicineName"
        outputVariable="newPrescription"/>
        <invoke partnerLink="pharmacyService" operation="getQuantity" outputVariable="quantity"/>
        <invoke partnerLink="pharmacyService" operation="getFrequency" outputVariable="frequency"/>
        <invoke partnerLink="pharmacyService" operation="getDuration" outputVariable="duration"/>
```

```
<invoke partnerLink="pharmacyService" operation="getOptions" outputVariable="options"/>
<invoke partnerLink="pharmacyService" operation="getComments" outputVariable="comments"/>
<assign>
        <copy>
                <from>Xanax</from>
                <to variable="oldPrescription"/>
        </copy>
</assign>
<assign>
        <copy>
                <from>milk</from>
                <to variable="food"/>
        </copy>
</assign>
<assign>
    <copy>
            <from>diabetes</from>
            <to variable="condition"/>
    </copy>
</assign>
<assign>
    <copy>
            <from>Congratulations, no conflicts has been found!</from>
            <to variable="noConflictString"/>
    </copy>
</assign>
<assign>
    <copy>
            <from>Warning, a conflict has been found!</from>
```

```
            <to variable="conflictString"/>
        </copy>
    </assign>
    <assign>
        <copy>
            <from>josemreyes@gmail.com </from>
            <to variable="email"/>
        </copy>
    </assign>
    <invoke partnerLink="mcdService" operation="checkMedicinesConflicts" inputVariable="newPrescription, oldPrescription" outputVariable="hasConflicts"/>
    <invoke partnerLink="mcdService" operation="checkConditionsConflicts" inputVariable="newPrescription, condition" outputVariable="hasConflicts" />
    <invoke partnerLink="mcdService" operation="checkFoodConflicts" inputVariable="newPrescription, food" outputVariable="hasConflicts"/>
    <if>
        <condition>
            hasConflicts
        </condition>
        <invoke partnerLink="speechService" operation="speak" inputVariable="conflictString"/>
        <invoke partnerLink="notificationService" operation="emailandtext" inputVariable="email, conflictString"/>
    <else>
        <invoke partnerLink="speechService" operation="speak" inputVariable="noConflictString"/>
        <invoke partnerLink="notificationService" operation="emailandtext" inputVariable="email, noConflictString"/>
    </else>
    </if>
    <invoke partnerLink="medSHService" operation="setTag" inputVariable="rfidTag"/>
    <invoke partnerLink="medSHService" operation="setFirstName" inputVariable="firstName"/>
    <invoke partnerLink="medSHService" operation="setLastName" inputVariable="lastName"/>
```

```
    <invoke partnerLink="medSHService" operation="setMedicineName" inputVariable="newPrescription"/>
    <invoke partnerLink="medSHService" operation="setQuantity" inputVariable="quantity"/>
    <invoke partnerLink="medSHService" operation="setFrequency" inputVariable="frequency"/>
    <invoke partnerLink="medSHService" operation="setDuration" inputVariable="duration"/>
    <invoke partnerLink="medSHService" operation="setOptions" inputVariable="options"/>
    <invoke partnerLink="medSHService" operation="setComments" inputVariable="comments"/>

    <invoke partnerLink="medSHService" operation="updateMedicinesDB"/>
    </sequence>
</process>
```

APPENDIX C. MISS Model in PROMELA

```
typedef covered_entity {

/* This record will define the general categories of the different requirements that a covered entity must fulfill in order to obey the HIPAA law.

*  Boolean values are used to represent this compliance or lack of it.

*  Type indentifies the category to which the the requestor belongs: 1- individual; 2- covered entity; 3- government related  */

      byte type;

      bool patientReqs;

      bool adminReqs;

}


typedef patient_record {

/* Boolean values represent what data exits in the covered entity's record and what is being forwarded. */

      bool patient_data;

      bool restricted_data;

}

/* Definition of the channels to be used for transmiting the data */

chan mcdChannel = [5] of {byte, bool};

chan startDoctor = [5] of {bool};

chan doctor2pharmacy = [5] of {byte, bool};

chan pharmacy2sh = [5] of {byte, bool};

chan requestDoctor = [5] of {byte, patient_record};

chan requestPharmacy = [5] of {byte, patient_record};

chan requestSH = [5] of {byte, patient_record};

/* Definition of the different convered entities. Declared global to check for compliance */

covered_entity doctorCE;

covered_entity pharmacyCE;

covered_entity shCE;
```

```
/*Variable to track succesful/unsuccesful execution of subsystems */
bool doctorFinished;
bool pharmacyFinished;
bool shFinished;
/*Variables for MCD */
byte category;
bool response;
/*Variables for RequestData for tracking purposes*/
byte requestor;
bool patientData;
bool restrictedData;
/*Variables for timeliness, completeness and notifications */
bool timeliness;
bool completeness;
bool notification;
active proctype mcd (){
/* Receive the prescription non-deterministically send true/false */
	bool message;
	mcdChannel?category,message;
	if
	:: response = true;
	:: response = false;
	fi;
	mcdChannel!category, response;
}
active proctype doctor () {
	bool start;
	doctorFinished = false;
	doctorCE.type = 2;
```

```
if
:: doctorCE.patientReqs = true
:: doctorCE.patientReqs = false
fi;
if
:: doctorCE.adminReqs = true;
:: doctorCE.adminReqs = false;
fi;
patient_record prescription, rec;
prescription.patient_data = true;
if
:: prescription.restricted_data = true;
:: prescription.restricted_data = false;
fi;
if
::true;
      mcdChannel!1,prescription.patient_data;
      mcdChannel?category,response;
      if
      ::response ->
                  if
                  ::notification = true;
                  ::notification = false;
                  fi;
      ::else;
      fi;
      doctor2pharmacy!1,prescription.patient_data;
::requestDoctor?category,rec;
      requestDoctor!category,prescription;
```

```
        fi;
        doctorFinished = true;
}
active proctype pharmacy () {
        bool prescriptionData;
        pharmacyCE.type = 2;
        pharmacyFinished = false;
        if
        :: pharmacyCE.patientReqs = true
        :: pharmacyCE.patientReqs = false
        fi;
        if
        :: pharmacyCE.adminReqs = true;
        :: pharmacyCE.adminReqs = false;
        fi;
        patient_record prescription, rec;
        if
        :: prescription.restricted_data = true;
        :: prescription.restricted_data = false;
        fi;
        if
        ::doctor2pharmacy?category,prescriptionData;
                prescription.patient_data = prescriptionData;
                mcdChannel!2,prescription.patient_data;
                mcdChannel?category, response;
                if
                ::response ->
                                if
                                ::notification = true;
```

```
                ::notification = false;
                fi;
                ::else;
            fi;
            pharmacy2sh!2,prescription.patient_data;

        ::requestPharmacy::requestPharmacy?category,rec;
            requestPharmacy!category,prescription;
        fi;
        pharmacyFinished = true;
}

active proctype smart_home (){
        bool prescriptionData;
        shFinished = false;
        shCE.type = 2;
        if
        :: shCE.patientReqs = true
        :: shCE.patientReqs = false
        fi;
        if
        :: shCE.adminReqs = true;
        :: shCE.adminReqs = false;
        fi;
        patient_record prescription, rec;
        if
        :: prescription.restricted_data = true;
        :: prescription.restricted_data = false;
        fi;
```

```
/*Indicates the end of the MISS system model */
endMISS:
/* Receiving the prescription from the pharmacy's module */
if
::pharmacy2sh?category,prescriptionData;
      prescription.patient_data = prescriptionData;
      mcdChannel!2,prescription.patient_data;
      mcdChannel?category, response;
      if
      ::response ->
                  if
                  ::notification = true;
                  ::notification = false;
                  fi;
      ::else;
      fi;
  :requestSH?category,rec;
      requestPharmacy!category,prescription;
  fi;
if
      ::timeliness = true;
      ::timeliness = false;
fi;
if
      ::completeness = true;
      ::completeness = false;
fi;
if
      ::notification = true;
```

```
            ::notification =false;
        fi;
        shFinished = true;
}
active proctype requestData(){
        patient_record record;
        do::
         if
            : requestor = 1;
            :: requestor = 2;
            :: requestor = 3;
        fi;
        if
        :: requestDoctor!requestor,record ->
            requestDoctor?requestor,record;
        :: requestPharmacy!requestor,record ->
            requestPharmacy?requestor,record;
        :: requestSH!requestor,record ->
            requestSH?requestor,record;
        fi;
        patientData = record.patient_data;
        restrictedData = record.restricted_data;
        od
}
init {
        run mcd();
        run doctor();
        run pharmacy();
        run smart_home();
```

```
    run  requestData();
}
```

APPENDIX D. MISS Model in oWFN

```
{
        net size:     |P|=50, |P_in|= 12, |P_out|= 24, |T|=13, |F|=62
}


    PLACE
     INTERNAL
      p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14;

     INPUT
      in.mcdService.checkConditionsConflicts {$ MAX_OCCURRENCES = 1 $},
      in.mcdService.checkFoodConflicts {$ MAX_OCCURRENCES = 1 $},
      in.mcdService.checkMedicinesConflicts {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getComments {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getDuration {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getFirstName {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getFrequency {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getLastName {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getMedicineName {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getOptions {$ MAX_OCCURRENCES = 1 $},
      in.pharmacyService.getQuantity {$ MAX_OCCURRENCES = 1 $},
      in.rfidService.getRFIDTag {$ MAX_OCCURRENCES = 1 $};

     OUTPUT
      out.mcdService.checkConditionsConflicts {$ MAX_OCCURRENCES = 1 $},
      out.mcdService.checkFoodConflicts {$ MAX_OCCURRENCES = 1 $},
      out.mcdService.checkMedicinesConflicts {$ MAX_OCCURRENCES = 1 $},
      out.medSHService.setComments {$ MAX_OCCURRENCES = 1 $},
```

```
   out.medSHService.setDuration {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setFirstName {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setFrequency {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setLastName {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setMedicineName {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setOptions {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setQuantity {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.setTag {$ MAX_OCCURRENCES = 1 $},
   out.medSHService.updateMedicinesDB {$ MAX_OCCURRENCES = 1 $},
   out.notificationService.emailandtext {$ MAX_OCCURRENCES = 2 $},
   out.pharmacyService.getComments {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getDuration {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getFirstName {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getFrequency {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getLastName {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getMedicineName {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getOptions {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.getQuantity {$ MAX_OCCURRENCES = 1 $},
   out.pharmacyService.setTag {$ MAX_OCCURRENCES = 1 $},
   out.rfidService.getRFIDTag {$ MAX_OCCURRENCES = 1 $};


INITIALMARKING
  p2:1 {initial place};

FINALMARKING
  p1 {final place};

TRANSITION t1 { output }
```

```
  CONSUME p2;
  PRODUCE out.rfidService.getRFIDTag, p3;

TRANSITION t2 { input/output }
  CONSUME in.pharmacyService.getLastName, p5;
  PRODUCE out.pharmacyService.getMedicineName, p6;

TRANSITION t3 { input/output }
  CONSUME in.pharmacyService.getQuantity, p7;
  PRODUCE out.pharmacyService.getFrequency, p8;

TRANSITION t4 { input/output }
  CONSUME in.pharmacyService.getFrequency, p8;
  PRODUCE out.pharmacyService.getDuration, p9;

TRANSITION t5 { input/output }
  CONSUME in.pharmacyService.getDuration, p9;
  PRODUCE out.pharmacyService.getOptions, p10;

TRANSITION t6 { input/output }
  CONSUME in.pharmacyService.getOptions, p10;
  PRODUCE out.pharmacyService.getComments, p11;

TRANSITION t7 { input/output }
  CONSUME in.pharmacyService.getComments, p11;
  PRODUCE out.mcdService.checkMedicinesConflicts, p12;

TRANSITION t8 { input/output }
  CONSUME in.mcdService.checkMedicinesConflicts, p12;
```

PRODUCE out.mcdService.checkConditionsConflicts, p13;

TRANSITION t9 { input/output }

CONSUME in.pharmacyService.getMedicineName, p6;

PRODUCE out.pharmacyService.getQuantity, p7;

TRANSITION t10 { input/output }

CONSUME in.mcdService.checkConditionsConflicts, p13;

PRODUCE out.mcdService.checkFoodConflicts, p14;

TRANSITION t11 { input/output }

CONSUME in.mcdService.checkFoodConflicts, p14;

PRODUCE out.medSHService.setComments, out.medSHService.setDuration, out.medSHService.setFirstName, out.medSHService.setFrequency, out.medSHService.setLastName, out.medSHService.setMedicineName, out.medSHService.setOptions, out.medSHService.setQuantity, out.medSHService.setTag, out.medSHService.updateMedicinesDB, out.notificationService.emailandtext, p1;

TRANSITION t12 { input/output }

CONSUME in.pharmacyService.getFirstName, p4;

PRODUCE out.pharmacyService.getLastName, p5;

TRANSITION t13 { input/output }

CONSUME in.rfidService.getRFIDTag, p3;

PRODUCE out.pharmacyService.getFirstName, out.pharmacyService.setTag, p4;

{ END OF FILE }