SERVICE ORIENTED DEVELOPMENT THROUGH AXIOMATIC DESIGN

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EBRU KULOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2011

Approval of the thesis:

SERVICE ORIENTED DEVELOPMENT THROUGH AXIOMATIC DESIGN

submitted by **EBRU KULOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Ali Hikmet Doğru
Supervisor, **Computer Engineering Dept., METU** _____

Assist. Prof. Dr. Cengiz Toğay
Co-supervisor, **Computer Eng. Dept., Çanakkale Onsekiz Mart Uni.** _____

Examining Committee Members:

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Ali Hikmet Doğru
Computer Engineering Dept., METU _____

Dr. Cevat Şener
Computer Engineering Dept., METU _____

Ediz Acar
Senior Expert Engineer, ASELSAN _____

Oğuz Özün
Senior Expert Engineer, ASELSAN _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: EBRU KULOĞLU

Signature :

ABSTRACT

SERVICE ORIENTED DEVELOPMENT THROUGH AXIOMATIC DESIGN

Kuloğlu, Ebru
M.Sc., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Ali Hikmet Doğru
Co-Supervisor : Assist. Prof. Dr. Cengiz Toğay

February 2011, 61 pages

This research combines the methodology proposed in Axiomatic Design Theory (ADT) with a service oriented decomposition approach for systematic development of Service Oriented Architecture compliant systems. A previous study had applied ADT to component oriented development where simultaneous specification and decomposition of models related to requirements, design, product domain, and components were supported. Recently, Web services have gained popularity and they became a more desired alternative to components. This research sets the foundation for service-oriented modeling and development with ADT support through enhancing the component oriented work conducted before. The goal is to be able to consider customer needs viewed in the domain context, together with the requirements and design so that efficient development can take place based on existing Web services. The system under development is viewed as a hierarchy of process models where leaf-level processes correspond to Web services.

Keywords: Axiomatic design, service-oriented architecture, web service

ÖZ

AKSİYOMATİK TASARIM İLE SERVİS YÖNELİMLİ GELİŞTİRME

Kuloğlu, Ebru
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Ali Hikmet Doğru
Ortak Tez Yöneticisi : Yrd. Doç. Dr. Cengiz Toğay

Şubat 2011, 61 sayfa

Bu çalışmada, aksiyomatik tasarım metodolojisi servis yönelimli yazılım mimarisiyle uyumlu sistemlerin tasarımı için kullanılacaktır. Daha önce bileşen yönelimli yazılım mimarisinde aksiyomatik tasarım metodolojisinin uygulandığı bir çalışma yapılmıştır. Bu çalışma kapsamında, müşteri isterleri, ürün alan tanımı ve bileşen kavramları desteklenmiştir. Günümüzde, web servislerinin popüleritesi artmış ve bileşenlere kıyasla daha fazla tercih edilecek hale gelmiştir. Bu araştırma, aksiyomatik tasarım desteğiyle servis yönelimli modelleme ve geliştirme esası, daha önce yürütülmüş bileşen yönelimli çalışmayla desteklenecektir. Hedef, ürün alanında sunulan müşteri isterlerini, gereksinimleri ve tasarımı birlikte düşünüp mevcut web servislerini etkin bir şekilde, geliştirme safhasında kullanmaktır. Çalışmada, süreç modelleri hiyerarşik bir yapıda sunulmakta, bu yapının yaprak seviyesindeki süreçleri de mevcut web servisleri ile karşılanmaktadır.

Anahtar Kelimeler: Aksiyomatik tasarım, servis yönelimli mimari, web servisi

To my Nephew Aybars Kuloğlu

ACKNOWLEDGMENTS

First of all, I would like to thank to my advisor Assoc. Prof. Dr. Ali Doğru, and co-advisor Assist. Prof. Dr. Cengiz Toğay, for their help, and technical support. They not only guided me, but also encouraged me throughout this study.

I would like to give my thanks to my family for giving me the heads up for getting my thesis done, and to my kitty cat Karamel.

I would also like to thank to Barış Karadeniz, for his technical support during my thesis period.

Last, but not the least, I would also like to thank to staff member in CENG department, Perihan İlgün for her willingness, and actual support with the procedures.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ v

ACKNOWLEDGMENTS vii

TABLE OF CONTENTS viii

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTERS

1 INTRODUCTION 1

2 BACKGROUND 3

2.1 Need for Decomposition: A Good Design 3

2.2 Axiomatic Design Theory Methodology 5

2.3 Service Oriented Approach 11

2.3.1 Service Oriented Architecture 11

2.3.2 Web Services 14

2.3.3 Web Service Orchestration 17

2.4 Business Process Execution Language 19

3 PROPOSED APPROACH: Service Oriented Architecture with Axiomatic Design 22

4 A CASE STUDY: DESIGNING AND MODELING A MILITARY DEPLOYMENT PLANNING SYSTEM 29

4.1 Domain Description 29

4.2 Software Analysis, System Design and Modeling 31

4.2.1 Introduction to Military Deployment Planning Software and Reverse Engineering 32

4.2.2 Web Service Design 34

4.2.3 Application Design 37
5 CONCLUSION AND FUTURE WORK 43
REFERENCES 46
APPENDICES
A ADSO DESIGN AND MODELLING TOOL 48
A.1 Adso 48
A.1.1 Defining a new Web Service into the Application Domain 49
A.1.2 Constructing the Application FR-DP Design Matrix and SOSEML representation of the Application 55
A.1.3 Modeling Processes in BPEL Designer 55

LIST OF TABLES

TABLES

Table 2.1 Primitive Activities 20

Table 2.2 Structured Activities 20

Table 2.3 Additional usages in BPEL 21

Table 4.1 Units and their corresponding task assignments. 31

LIST OF FIGURES

FIGURES

Figure 2.1 Axiomatic Design Procedures tailored for Service-Oriented Software Systems (The V Model) (Adapted from [4]) 6

Figure 2.2 Axiomatic Design Domains 7

Figure 2.3 Decomposition by zigzagging (adapted from [6]) 9

Figure 2.4 Probability distribution of a DP; solid line refers to uniform distribution, while dotted line refers to nonuniform distribution (adapted from [7]) 11

Figure 2.5 Web Services Architectural Model (Adapted from [9]) 12

Figure 2.6 Layers formed in SOA as the Web service orchestration evolved, and requirements arose. (Layers are numbered in chronological order corresponding to the improvements in SOA.) 13

Figure 2.7 Web Service Interface and Invocation 15

Figure 2.8 Orchestration, and Choreography of Web Services (Adapted from [12]) . 18

Figure 2.9 XLANG and WSFL-styles (Adapted from [13]) 19

Figure 3.1 Process, Web service, Web service interface, and link figure samples . . . 25

Figure 4.1 Design matrix for *Get all army corps inventory* Web service 35

Figure 4.2 SOSEML representation for *Get all army corps inventory* Web service . . 36

Figure 4.3 Web service list provided at any stage of designing study 36

Figure 4.4 Application Design Matrix for Military Deployment Planning Software . 38

Figure 4.5 Application SOSEML hierarchy tree for Military Deployment Planning Software 39

Figure 4.6 Application SOSEML hierarchy tree for Military Deployment Planning Software (continued) 40

Figure 4.7 Application SOSEML hierarchy tree for Military Deployment Planning Software (continued) 41

Figure 4.8 Application SOSEML hierarchy tree for Military Deployment Planning Software (continued) 42

Figure A.1 The pop up window to insert the newly defined Web Service name 49

Figure A.2 A partial screen shot to visualize the existing web services depicted on Web service design tab 50

Figure A.3 The abstract design matrix for the newly added Web service 51

Figure A.4 Demonstration of FR and DP details in the design matrix 52

Figure A.5 Adding a new FR-DP couple to the design matrix 53

Figure A.6 The Web Service FR-DP design matrix and corresponding SOSEML representation 54

Figure A.7 Design matrix with all FRs coupled by related DPs for the Application . . 56

Figure A.8 Level 1 and 2 decomposition of the Deployment Decision Support Application sample 57

Figure A.9 Level 1 and 2 decomposition of the Deployment Decision Support Application sample (continued) 58

Figure A.10 Level 1 and 2 decomposition of the Deployment Decision Support Application sample (continued) 59

Figure A.11 The Properties window for a process with no existing bpel file 60

Figure A.12 The BPEL Designer showing “introduce weapon properties”process model, the process figure on SOSEML tree has its BPEL icon light on, indicating the process has an existing bpel file 61

Figure A.13 The corresponding wsdl file for introduce weapon properties process model shown in BPEL designer. 61

CHAPTER 1

INTRODUCTION

Software systems have matured, and become well equipped with the improvement of a variety of architectures, one of which is Service Oriented Architecture (SOA) [26]. The component oriented approach has been followed by the service oriented approach, since the physical key components of service orientation, which are Web services, have supplied the developers with

- the standards such as WSDL, UDDI, SOAP, and HTTP;
- sufficient information contents in the Web service interfaces to explain the Web services capabilities and functionalities;
- easiness to reach Web services through internet;
- security policies;
- invocation standards defined in their corresponding interfaces in WSDL format, thus maintaining a control mechanism for no type problems; such as invoking the Web service with wrong calls is prohibited by the proposed mechanism [1].

Following a service oriented philosophy, we shall see that the tendency in offering approaches for service composition and integration cannot reach the maturity level reached in technological dimension of SOA [1]. Service oriented development is very important and is too much in demand. There are some mechanisms that support the development of large software intensive systems that exploit the advantages of web services. The technology stack accompanying SOA is a good example to such support. However, while the process modeling abstraction that comes within such environment and is shaped by BPMN and BPEL support the idea to hierarchically organize the decomposition of the solution; there is not enough guidance for

the methodology to decompose. A different approach had been instrumental in decomposing the requirements and design spaces concurrently and applied to Object Oriented development, was already adopted by component oriented approaches [14]. The missing support for a methodology to decompose the 'process' space so that a developer could start with the system specification and arrive at the existing web services in a top-down approach has been the motivation behind this thesis research. The capability to concurrently decompose the problem specification model and the solution, as proposed by the Axiomatic Design Theory (ADT) [3], is employed in this work to support development in the SOA world. This is why we have decided to study a decomposition technique, and a relevant modeling method for integration purposes on service oriented architectures. This thesis work includes 5 chapters. In Chapter 2, we have given necessary background on ADT, service oriented architecture, Web services, and business process management, and modeling tool BPEL. In Chapter 3, we have described our proposed approach on service oriented architecture with axiomatic design. In Chapter 4, we have provided a case study involving a previous study, and provided an extensive work through the Axiomatic Design in Service Orientation (ADSO) methodology. Finally in Chapter 5, we have concluded our thesis study, and have given our opinion on possible future work. We have also provided the implementation details and graphical user interfaces that would help the ADSO user to understand the concepts on the tool in the appendix section.

CHAPTER 2

BACKGROUND

In this chapter, we shall learn why, and how we could obtain a good design, then we will move onto the detailed explanations of the key concepts before we get introduced with our proposed methodology; which are Axiomatic Design Theory (ADT) methodology [4], service oriented approach, and Business Process Execution Language (BPEL). We will also be provided with sub topics in service oriented approach part, where we will be introduced with the Service Oriented Architecture (SOA), Web Service (WS), and Web Service Orchestration (WSO), Web Service Choreography (WS-Choreography) concepts.

2.1 Need for Decomposition: A Good Design

It has always been a hard work for developers to give support during software debugging, or software usage process. They sometimes may face a pile of code, getting messy with the code patches. An exponential mess growth in code while an incremental decline in time left until deadline may scare the developers more than anything. Another tragic case would be, main programmer quitting the job, and then the projects end up with being developed from scratch. These scenarios suggest software development being a science more than an art. It is believed that good design would lead to a more reliable software development process, giving less trouble to the developer. There are two approaches to consider a good design to be a budget controlling mechanism. If the customer needs are well formed, one should not hesitate to form the design proposed by ADT methodology. But once the requirements become blurry, there are again two ways to take according to what is in hand. One is, when the hardware that your software will be running on cost too much, then whether you have the services to accomplish the blurry requirements or not, your goal would be to quit designing the system

with the indefinite functional requirements, and to start developing a prototype of the system. Another situation is when the risk of failure does not cost a lot for the designer in this situation, if the services that are capable of accomplishing a variety of design parameters exist, then the designer would map those blurry requirements to the existing design parameters, and come up with some design matrices. Among these design matrices, the designer can pick the most appropriate matrix to design the system.

The software designers have leaned to finding some ways to understand whether a design is good or not. This concern has introduced us with the paradigm, the Decision Based Design (DBD) [2], and with especially the increase in demand on concurrent design support tools, the researches on DBD has expanded. Understanding the accuracy of design actually relies on following some validation rules. First of all your design must be logical, since it may be subjected to some changes in the future according to the customer profile change, or an extension, or change in customer needs. To be able to keep up with these changes, the results coming from the design model has to be intuitive. Understanding how much a design is logical is hard to infer, but a logical design can handle possible change requests in the future, so intuition is the way to evaluate this qualitative attribute. The second validation rule is the design should embrace the uncertainty, and provide reliable information. The field experts should help during the design process, so that would support the reliability of information content in the system. The design model should also be aware of uncertainties that would lead to possible errors in the achieved results. This would give the designer to have the confidence to expect, and courage to handle these results. Another validation rule to be considered is not interfering with the preferences of the designer. If the designer, utilizing the methodology, is forced to use a specific preference during the design process, this would result in an influence on the outcome. However, if the methodology supports the designer to use a set of preferences, this would give the designers the opportunity to be durable to changes that would occur over time, since the customers change their goals, and accordingly their needs, in order to stay in the market [2]. Baring these rules in mind, and considering the advantages, and disadvantages of possible design methodologies serve, we have decided to use axiomatic design approach during this study.

2.2 Axiomatic Design Theory Methodology

The ultimate goal of designs is to provide effective systems. These systems may be in engineering fields, as well as business, or government. The systems have been produced empirically, or intuitively, since designers have not been introduced with a formal and theoretical framework which would support the system design. Qualitative approaches have been used in design processes, and in order to weigh the robustness of the system, development, and even testing phases have been completed. These stages cost too much to the companies, and the results of the systems were unreliable, because they were processed with empirical decisions [3]. Axiomatic Design Theory (ADT) is a decision support methodology for design, developed by Dr. Suh Nam Pyo [3]. First ADT usage in software development had been proposed by Sung-He Do, and Nam Pyo Suh where they have developed the Acclaro [8] design software. They had also conducted a case study to apply axiomatic design to object oriented programming. Figure 2.1 had been first adapted for object-oriented software for Acclaro design software case study. Afterwards, axiomatic design has been applied to component-oriented approach [14], and the V-model has been adapted to show the top-down and bottom-up approach for component oriented systems. In this study, we have adapted the original V-model [8] into the V-model that serves for axiomatic design usage on service-oriented applications.

Software designs following ADT are self-consistent, and they contain uncoupled or decoupled inter-relationships and arrangements among the services. These designs are easy to change, in forms of omitting some parts, extending the design, or modifying parts within the design. ADT serves designers to make correct decisions during design process, and come up with the possibly accurate design resulting in these advantages [4]. ADT adapts a top-down approach, and decomposes the system into possible smaller grained pieces. The advantage of ADT is it provides the designer to make simultaneous decomposition. Axiomatic design methodology encapsulates four concepts. These are domains, hierarchy, zigzagging, and axioms. Domains are; costumer domain, functional domain, physical domain, and process domain, adapting customer needs (CN), functional requirements (FR), design parameters (DP), and process variables (PV) respectively.

Customer needs refer to what customer wants in ADT approach. How customer needs are accomplished is however in the responsibility of functional requirements. FRs are definitions of system requirements which aim to satisfy the customer expectations from the software

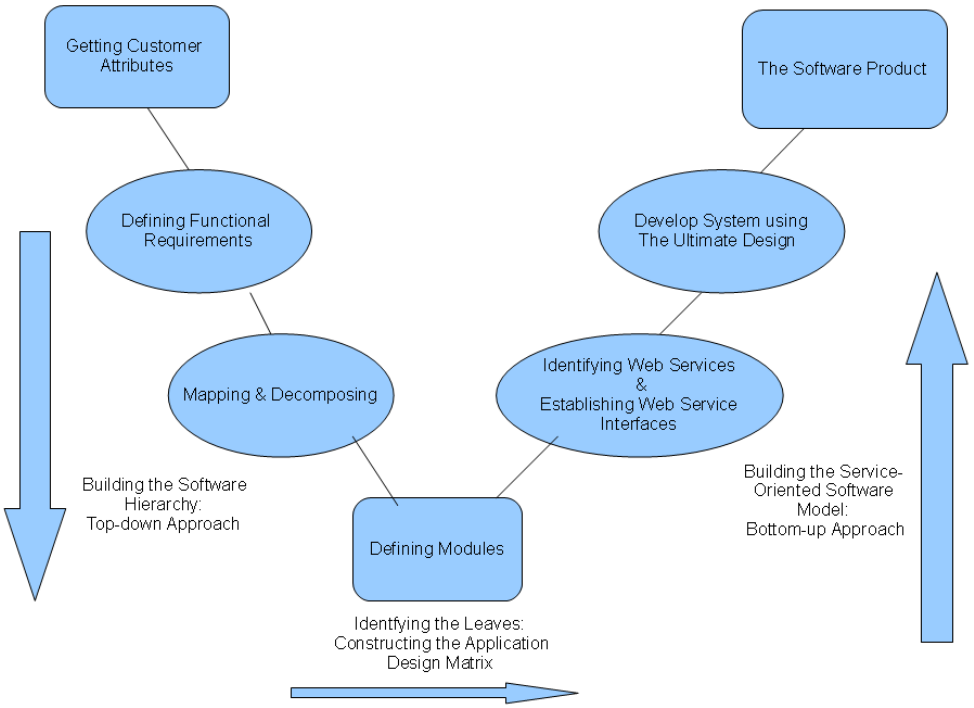

Figure 2.1: Axiomatic Design Procedures tailored for Service-Oriented Software Systems (The V Model) (Adapted from [4])

product, as shown in Figure 2.2. In other words, FRs should describe the expectations from the product. In order to define functional requirements, designer starts with the top system requirement, and decomposes this FR hierarchically. The ADT methodology offers the designer to decompose all four domains concurrently, in order to make use of existing DPs, and PVs. Concurrent decomposition continues until all leaf FRs in the FR hierarchy are handled with a corresponding DP. Let us illustrate the idea of decomposing FRs, and DPs with a real life example. Imagine that we have an application providing the appropriate seaside holiday locations in between the given arrival and departure dates. This means that we have a corresponding web service in our domain, that satisfies this application. On the other hand, let us assume that the customer requested an application that provides her with the best period of time to plant carrots on her land. This customer need does not have a corresponding design parameter in our domain. This is why, after we embody the customer need into our functional requirement, we begin to decompose the problem into finer functional requirements, and try to find a corresponding design parameter by the zigzagging approach in ADT. We have a big design parameter that satisfies the request for finding cities for seaside holiday for the period in between the dates the customer provided. When this web service is decomposed, we shall

see various finer web services that satisfy various types of requests, one of which would be requesting the forecast website for the weather for the user given dates. This web service also satisfies the customer need for best time to plant carrots, since she would be in need of the weather report for each day of the year. Our domain would provide us with the weather web service when our seaside holiday application is decomposed into finer design parameters.

Design parameters are defined as the key physical variables, and process variables are the corresponding implementations of design parameters. In our methodology, the process variables correspond to Web services, while process variables are interfaces of these Web services. ADT follows a V-model, as in Figure 2.1.

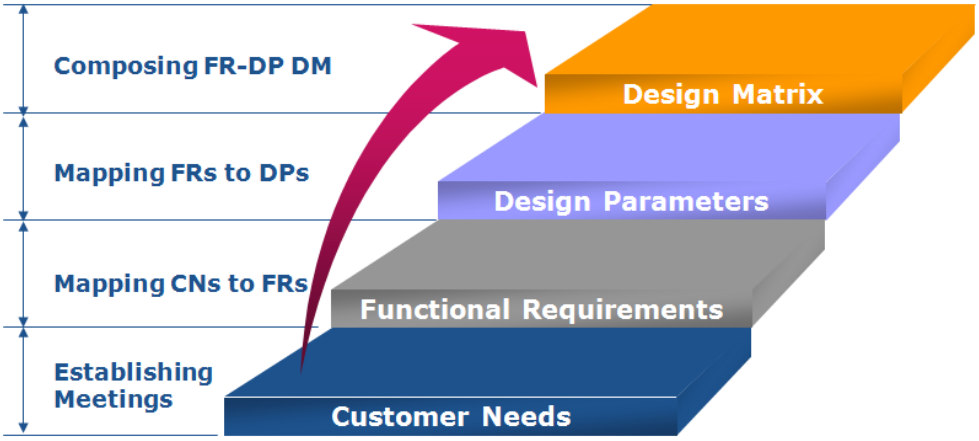

Figure 2.2: Axiomatic Design Domains

The model starts with the top-down approach, and then the finer grains are composed to fulfill the full system. Top-down procedure starts with getting the customer attributes, where customer defines their needs from the system to be designed. These attributes are called customer needs in the axiomatic design methodology. Their needs allow us to define the FRs of the system. Matching the CNs with FRs is type of a translation from customer world to design world. While the customer needs are being formed, the designer tries to come up with a corresponding functional requirement from the function domain. If designer provides a previous work with functional requirements, which is connected to the same domain with the customer's problem, then the customer would have an advantage to see different aspects that they might have missed, but realized via the provided requirements. After customer needs are matched with the functional requirements, designer concentrates on finding appropriate

design parameters for functional requirements. This mapping task is the following step of defining FRs in the V-model hierarchy. During this mapping, some FRs would find their design parameter match in designers' design domain, but some FRs might not find any ready-to-use DPs. Once the DPs are chosen, the designers are supposed to go to the process domain and identify the PVs. These PVs are either existing processes, or are to be introduced to the domain as new processes. At this point, designers may follow different behaviors. The FRs might be reviewed and reshaped according to the existing DPs. They may be combined, or decomposed, following the sense that CNs are still satisfied. Other option would be defining new DPs in accordance to the FRs given. These DPs are then supposed to be implemented, and they would appear in designers' process domain. As we proceed in the axiomatic design methodology, we actually decompose the system into smaller, affordable units while trying to map the FRs with DPs. The mapping procedure is followed among DPs and PVs as well. The interfaces are implemented, and exist in the process domain, ready to be composed for the whole system at the end of the design procedure. Design parameters are defined as the interfaces to the real methods. In component-oriented world, these are the component interfaces [14], while in the service-oriented approach as we will explore in this study, the DPs are provided as Web service interfaces. The process continues with a back and forth manner between functional domain and physical domain. We pick a functional requirement from the hierarchy we have built, then zig to the physical domain and try to find a corresponding design parameter.

Although there is a chance that design parameter exists in our library, there is the probability of not finding any matching DP for the given FR. Then we have to populate a related DP, and we implement it to get our PV. After we pick a proper DP from the design domain, we go back to functional domain, in ADT terminology this attempt is called zagging, we make a link in between the previous design parameter and next functional requirement in the sub level of functional requirement hierarchy.

This decomposition will occur until the design can be implemented without further decomposition to create PV, DP, and, FR hierarchies. The decomposition of three domains cannot be achieved by remaining in a single domain, but through zigzagging between these domains, [3]. The procedure till now has supported the designer with the simultaneous decomposition chance, 2.3. And, this simultaneous decomposition is preferred in order to provide the functional requirements with the existing design parameters by considering the decomposition

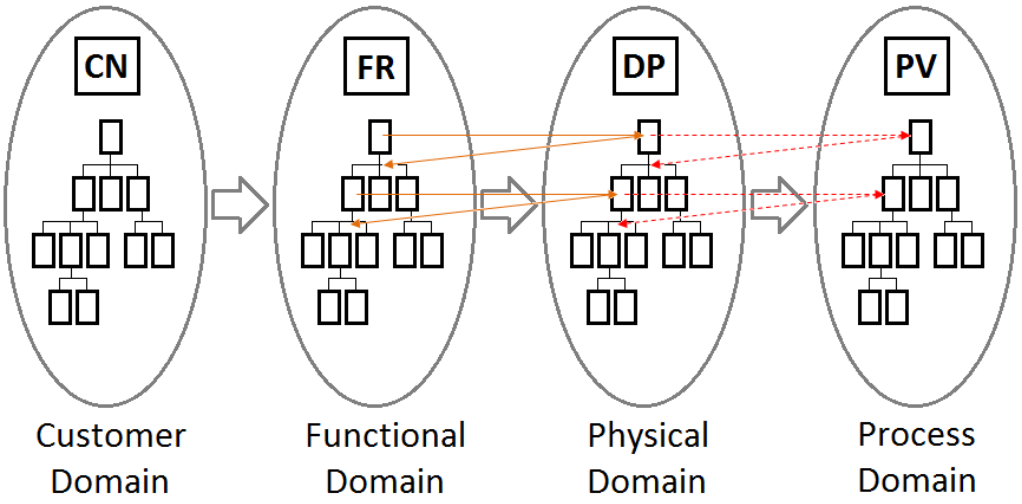

Figure 2.3: Decomposition by zigzagging (adapted from [6])

task not only in a single domain, but in multiple domains. In ADT, the relationship among FRs and DPs are shown in a design matrix. The PV, DP, and, FR hierarchies and their corresponding FR-DP, and DP-PV design matrices form the system design. The rows of this matrix are functional requirements, while the columns correspond to the design parameters. If there is a relation in between, we put an X in the related cell in the matrix, if not we put a 0. From now on, the purpose of the designer is to meet the two axioms ADT asserts; independence axiom, and information axiom. These axioms are useful designing tools, providing analytical measures. Independence axiom states that the functional requirements should be independent from each other. The relation between an FR and a DP is stated in the design matrix, and this correspondence shall form a square design matrix. In case of FR count exceeding DP count, this means that the FRs are not satisfied, or the design is coupled. If DP count exceeds FR count, then this would mean that the solution contains redundant functionality, and in both cases, the design matrix is not a square matrix [5]. If the FRs are not independent, then they have to be edited by for example, decomposing the requirements, or changing the content of a requirement, or gathering some requirements. The coupling among FRs are figured out on the design matrix, and stated as an uncoupled, decoupled, or coupled matrix. The ideal design is the uncoupled case, but it is very rare to reach this type of design. In an uncoupled design matrix, a diagonal design matrix is formed, meaning that each FR is satisfied by exactly one DP. In such a design, the system processes can be developed concurrently, run in parallel, since there is no coupling, thus no dependency among DPs. In this case, where all FRs can

be independently satisfied, a single process model, including these entire FRs can be placed in the corresponding domain for future use. In a decoupled design matrix however, the design matrix is triangular. This means that a sequence exists. In order satisfy all FRs, the DPs should be adjusted in a certain order. The last possible form of a design matrix is a coupled design. In such a case, the matrix is mostly consisted of nonzero elements. This design cannot suggest an independent solution for the FRs. A coupled design can be converted to a decoupled design, as [5] stated, but it comes at a price. An example solution for converting a coupled design to a decoupled one can be gathering some services together to satisfy some specific functionality. Although independence axiom is satisfied, in case of an addition of a new FR to the system, if the FR set already contains an FR close to the new FR in definition, the old FR can be replaced by a new FR satisfying both FRs, or a completely different set of FRs can be selected. In case of such a change in the FR domain, the previous DPs in the design solution cannot meet the requirements of the new FR set, so a new design solution must be pursued. The information axiom states that the DPs in the design should contain the least information possible. This axiom actually provides a mathematical approach to the design matrix. We are provided the opportunity to decide among the designs satisfying the independence axiom by comparing the information content of design matrices. For the purpose, we are provided an equation [7] for information axiom.

$$I = log(\frac{systemrange}{commonrange}) \tag{2.1}$$

As stated in Figure 2.4, there exists three terminologies to explain the variance capability of DPs in different domains. System range is the capability of the system to satisfy the FRs, while design range is the variation tolerance for DPs, and the common range is the overlap range between design and system ranges, where the FRs can be met. The design is a better design, if the equation of information axiom result is closer to zero. This means that the DPs are probabilistically independent, and the information content is the sum of the information content of all DPs in the design matrix. Thus, system range and common range collapse, and the result of the equation of Information Axiom is *log1*, which is *zero*. Since an uncoupled design is the ideal case, Suh proposed another method [6] to compute the information content of decoupled designs. However, there is not an exact method to compute the information content of coupled designs, and is thus left out of scope.

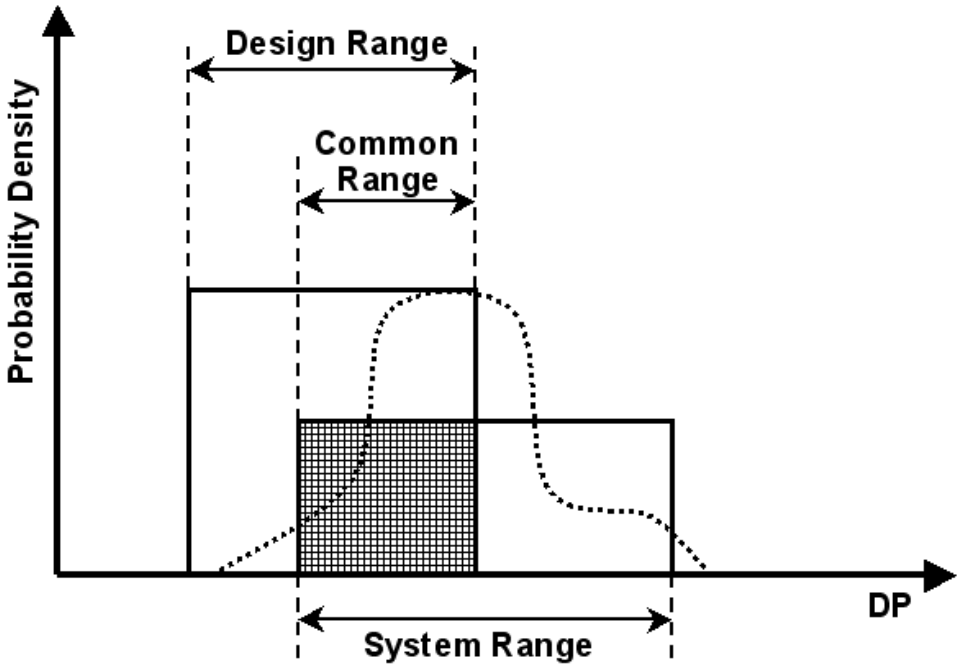

Figure 2.4: Probability distribution of a DP; solid line refers to uniform distribution, while dotted line refers to nonuniform distribution (adapted from [7])

2.3 Service Oriented Approach

During this section, we provide extensive information on Service Oriented Architecture (SOA), Web Service (WS), and Web Service Orchestration (WSO).

2.3.1 Service Oriented Architecture

Service orientation utilizes services as the constructs to support low-cost, easily composed, and rapid development of distributed applications. Services are computational units that are autonomous, and platform independent. Services can perform functionalities ranging from basic tasks to sophisticated business processes. They reflect a service oriented approach to programming with their nature of being discoverable and invokable through the network for compositional purposes, instead of building new applications.

The basic concepts in Service Oriented Architecture (SOA), as shown in Figure 2.5 are composed of the basic services being published, subscribed, and searched via the standard pro-

tocols that are already available at the market. In a typical service based scenario, basic interactions involving the description, publishing, finding, and binding of services are handled by three main parts. One is the service consumer, who finds the service registries, discovers a service endpoint, and retrieves the service description from the registry, or directly from the service provider through meta data exchange. The service description is used to either bind with the service provider, and invoke the service, or interact with the particular service. The second contributor is the service provider, as the name implies the provider of the service, defines a service description of the service and published it to the service consumer, or a service discovery agency. By publishing the service description to the consumer, or agency, the service is made discoverable the last contributor is the service linker/ aggregator, which provides the communication of the service consumer, and the provider.

In SOA, functionality is provided with not a monolithic application, but with the orchestration of several services. It is both applicable in software environments, and business industries, since it offers a constructional mechanism for services running at different parties.

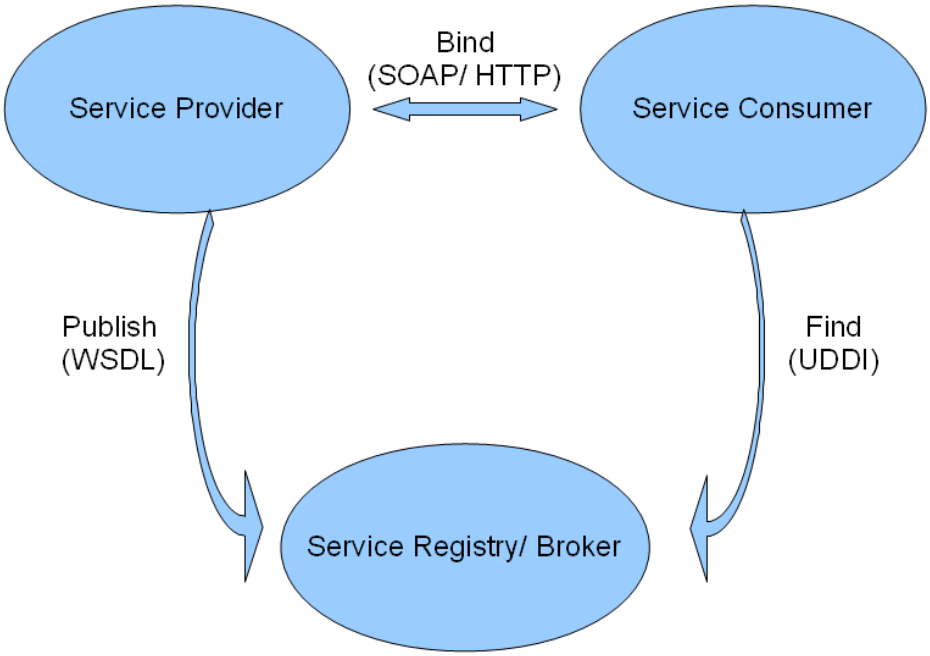

Figure 2.5: Web Services Architectural Model (Adapted from [9])

After accomplishing the orchestration of basic services, with the help of monitoring, suitability checking, and orchestration, and choreography methodologies, composite services were

introduced. They are the compositional structures consisting of the basic Web services, and serve as a Web service themselves.

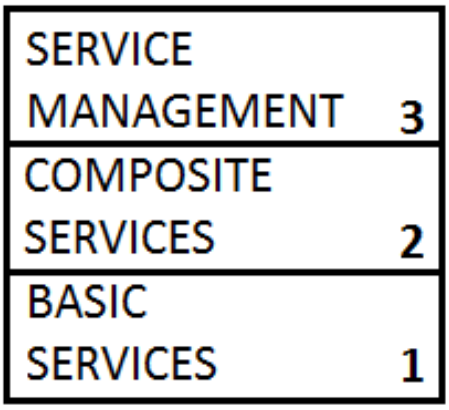

Figure 2.6: Layers formed in SOA as the Web service orchestration evolved, and requirements arose. (Layers are numbered in chronological order corresponding to the improvements in SOA.)

As the Web service coordination has evolved, new requirements in service management arose, such as assurance, service rating, and certification services. The health of the applications has to be constantly monitored since the additions to already existing components may overload the system, and cause a failure, bringing down many interdependent enterprise applications. Such an effect may also occur in case of changes in application components. The service management layer provides us with a variety of activities, ranging from installation and configuration to collecting metrics and tuning to ensure responsive service execution. Service level agreement negotiation, management, and auditing, monitoring, troubleshooting, service state management, performance management, and so forth are the rest of the activities service management layer provides.

The logical service-based architecture is known as extended Service Oriented Architecture, [17]. As depicted in Figure 2.6, the architectural layers in extended SOA provide a logical separation of functionality. This serves for the need to separate basic services provided by services middleware infrastructure and conventional SOA from advanced service functionality for dynamic composition of services. This extended SOA also lets us distinguish the functionality for composing services from the functionality of management of the services, [19].

Conventional development methodologies like object oriented development (OOD), and com-

ponent based development (CBD), can only address some requirements of service oriented computing applications. Services are subject to continuous maintenance and improvement in scope and performance, so that they can catch up with increasing number of consumers. Components are merely distributable objects, still they carry with them the difficulties of object modeling, increasing the scale of the model, and yet multiplying the complexity. Components also do not allow reuse and dynamic behavior as much as services do. Service oriented approach on the other hand serves an inter-disciplinary approach. SOA lets OOD and CBD contribute to general software architecture principles; information hiding, separation of concerns, and modularization, while business modeling in SOA helps in analysis of structuring of value-added-chains and improvement of processes [19]. The business modeling also helps workflow implementations being tested via defining how a business functions before they are designed and implemented. In other words, SOA fuses elements of OOD, and CBD with the elements of business modeling.

2.3.2 Web Services

Web service technology is an important realizing technology for SOA, and they act as the servers of today's developers [11]. Web Service (WS) provides the fulfillment of functionality, and satisfies applications, and businesses.

WS are accessed, as stated for SOA, through their interfaces by the service consumers, and express themselves in a standardized way, so that WS invocation can be achieved via using this standard structure. For this purpose, Web Service Definition Language (WSDL) serves Web services for defining their interfaces in a standardized way that WS can be invoked via internet protocols, and Simple Object Access Protocol (SOAP) to access them through internet protocols, as shown in Figure 2.7.

The ultimate goal of WS has been application integration since the first day. First, it only dealt with Enterprise Architecture Integration (EAI). The Web service technology was limited to the field of data and application integration. Then, its concern has shifted towards integrating business processes, and thus Web services got involved in Business to Business (B2B) environment. This caused the concern to become, the integration of Web services to business processes of the market. As stated in the case study given in [10], ideally, according to the demand on new business services, a comprehensive study is made on the preexisting business

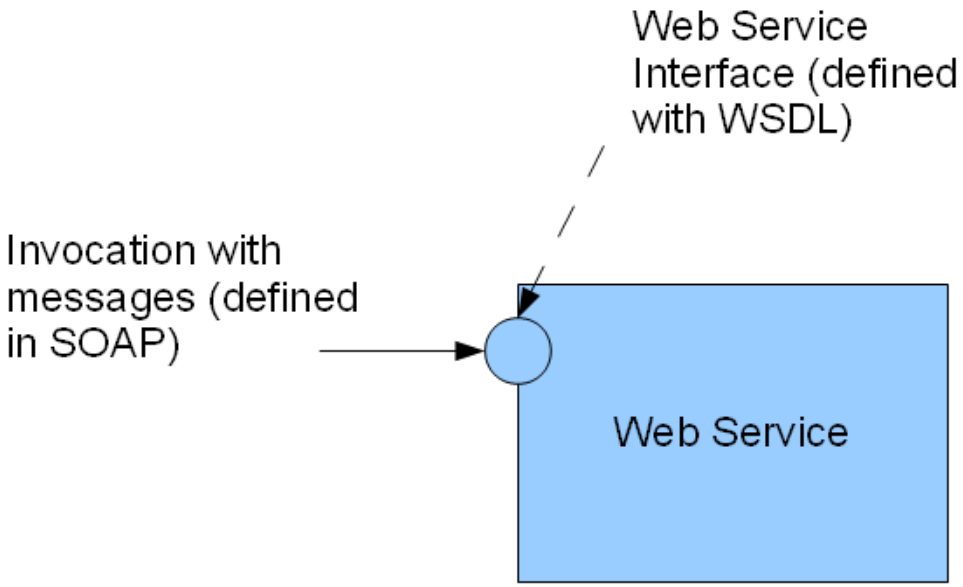

Figure 2.7: Web Service Interface and Invocation

services, and new business services are supplied instead of implementing new applications. In other words, the ultimate concern of Web services has been shifted towards achieving the integration of applications and business processes with existing process integration model. The orchestration of these preexisting business services are to be achieved on the fly, without implementing the integration process. According to this need, support for changes on demand, legacy changes, or reusability of functionality among services are to be covered in business process orchestration. To be more precise on expectations from Web service orchestration, we need to understand what additional requirements would arise if we would like to use this technology in business field. As stated in [10], here are some issues collected according to a case study done on governmental Web service technology usage:

- Reusable components and shared services: Participants using the same functionality could be provided a mechanism, where the functionality can be shared, or borrowed. This not only causes a decrease in effort for developing the functionality, but also a faster response time for changes in legacy, or customer needs. This spoken issue would induce a central mechanism to handle maintenance, as well as control, and update of available services without duplicating the effort for development in each participant.
- Information sharing aspects: The participants, namely actors in [10], have different

kinds of data in their registry. The orchestration mechanism should provide the ability to share this information among the participants. Then the data repository role would be maintained by one participant, and other participants would publish this Web service supplier for necessary information. For example for personal information validity checking Web service, information supplying service is needed. Another issue here is the planning, which Web service leads the other. This control mechanism has to be maintained by one participant, which means Web service management necessity occurs. The other issue that must be considered is the privacy of the information. If some information is not relevant to some parties, then that information should not be shared with the uninvolved participant.

- Accountability and responsibility: Allocation of responsibilities is an important issue for knowing who to apply for in case of a failure during the progress. This issue cannot be controlled by the Web service orchestration mechanism, so precautions must be taken.

These statements prove that although Web service technology is satisfying in Web service orchestration, some precautions for the organizational issues stated above has to be considered before Web service technology is applied to governmental usage.

On the other hand, Web services are supported by all major software vendors. So the usage of Web service technology is hard to reject. They are the first technology to promise universal inter-operability among applications effectively and widely, running on different platforms. They can use standard internet protocols, such as HTTP (Hyper Text Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and FTP (File Transfer Protocol), the communication among participants having Web service interactions. This universal inter-operability is achieved via some standards like, SOAP, WSDL, and Universal Description, Discovery and Integration (UDDI). These standards are written in XML, so messaging, and descriptions are easy to understand. These standards make Web services appropriate for system integrations, but further adaption of Web services require higher level standards. The need for consuming, or publishing services, Web service orchestration is needed. To be able to provide a promising solution to the problem of coordinating cross departmental processes, combination of Web services and Web service orchestration, [11], which will be described in the next section, are recommended to be utilized in the construction of service-oriented compliant systems.

2.3.3 Web Service Orchestration

To be able to utilize Web services for constructing the big system, we have to organize them in the way that the result is the desired system. Business process orchestration had been used in human activated process coordination before the advent of Web services, and with the introduction of Web services, business process orchestration has extended, or it is more correct if we say it shifted its terrain towards Web service integration. The key goal of Web service integration has been application integration. This all started and was limited with the Enterprise Application Integration, then Web services were used for Business to Business integration. In order to use Web services in B2B integration, Web Service Orchestration (WSO), which is the coordination of executable business processes, was developed. However, it is stated in [23] that, the full potential of Web services as an integration platform will be achieved when the application and business processes integrate their complex integrations via using a standard process integration model. The shift towards on the fly integration has introduced us with the two concepts; WSO and the other form, WS-Choreography. These two concepts are often used in place of each other by mistake. The distinction between WSO and WS-Choreography is on the abstraction level of coordinated processes. As we have stated, WSO is the coordination of executable business processes, while WS-Choreography is of abstract business processes. In other words, if coordination of Web services is achieved via executable business processes, this is called WSO. Executable business processes are involved in the execution order of constituent activities, the partners that are involved in the message exchange within the system, and the fault tolerance activities. On the other hand, abstract business processes, only explain the messaging mechanism among the involved participants, and not mention the detail about the mechanisms running in any of these participants. WSO is an application integration technology that is concerned about a single participant, and what happens at that particular participant is the business of WSO. However, WSO is not widely used in businesses and government, since its advantages have not been studied enough, and its newness in the software environment is thus undeniable. Let us introduce you with some advantages of WSO that has been reached by the research conducted in [11]. WSO has to be dynamic, flexible, and adaptable to change, in order to meet the changing business needs. First of all, WSO provides some standards that allow the developers to have less skill sets. Additionally, WSO also narrows the gap between business analysts, and software developers. WSO provides portability, and re-use of processes, and with the help of open standards, these

capabilities reduce the implementation costs. WSO also reduces the amount of time to deliver applications, and provide better maintainability, as well as less maintenance costs. The ability to deliver Information Systems (IS) with reusable software components allow better flexibility to the software companies, as well as freeing customers from choosing an all-in-one solution. In order to view a categorized table on the advantages of WSO, we may refer to [24].

WS-Choreography is dealing with the flow mechanism occurring among service supplier, and demander participants. A visual has been supplied in Figure 2.8 to clear out the distinction between WSO, and WS-Choreography.

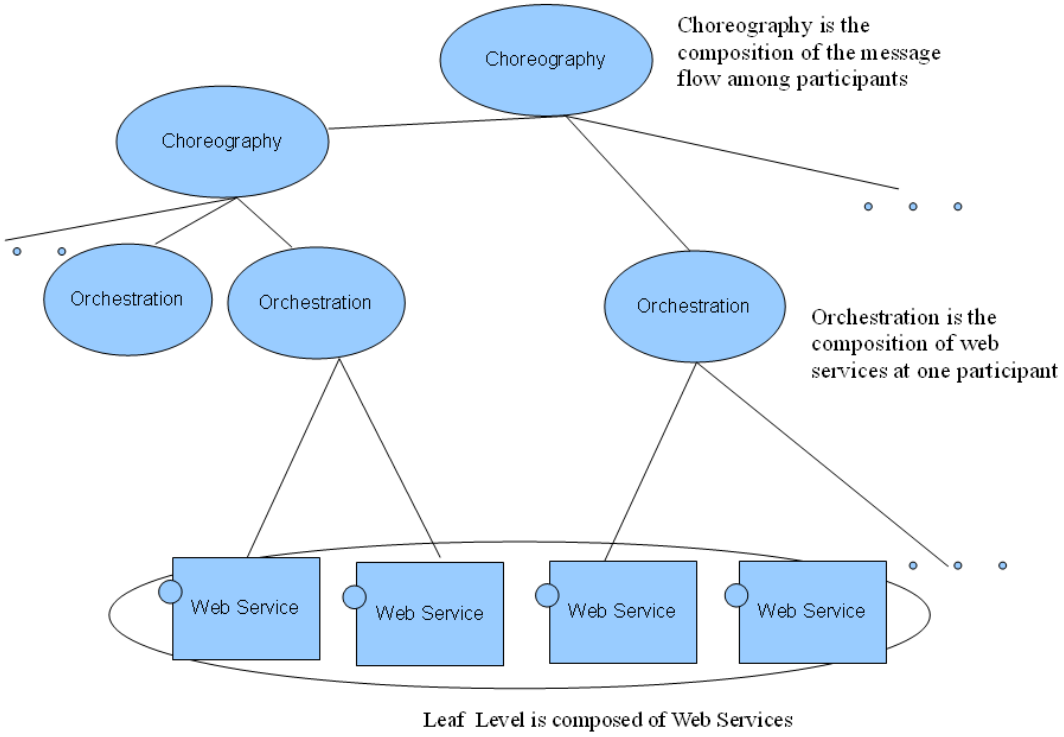

Figure 2.8: Orchestration, and Choreography of Web Services (Adapted from [12])

The knowledge of Web services used in a particular participant, the flow of these Web services is not a concern for WS-Choreograph. The process flow followed in the subscriber party to obtain the advice, or the time it takes is not a concern of the consumer party. WS-Choreography suggests that once a party asks for an advice to the other, the only responsibility of the subscribed party is to respond to the consumer with the advice. Generally speaking, WS-Choreography is involved in describing the message flow that can occur between service agencies.

2.4 Business Process Execution Language

Business Process Execution Language for Web Services (BPEL4WS), or Business Process Execution Language (BPEL) in short, is introduced, and developed by IBM, Microsoft, and BEA [25]. BPEL is the standard language to be used for realizing WSO, but it also contains a part for abstract business processes. BPEL is a kind of flow chart that models the behavior of Web services in a business process interaction [22], providing and XML-based grammar. BPEL coordinates Web services participating in a process flow via describing their control logic [21]. BPEL is a layer that is on top of WSDL, and the collaboration is provided such as, WSDL interface defines the specific operations allowed, while BPEL defines how to sequence those specific operations. In WSDL; every BPEL processes' entry and exit points are described. WSDL also provides data types to describe the information passing among process requests. BPEL processes' external source needs are also provided via WSDL's capability to reference external services [21]. BPEL is the combination of two technologies, IBM's WSFL, and Microsoft's XLANG. BPEL gets its block-structured language from XLANG, and graph-based language from WSFL.

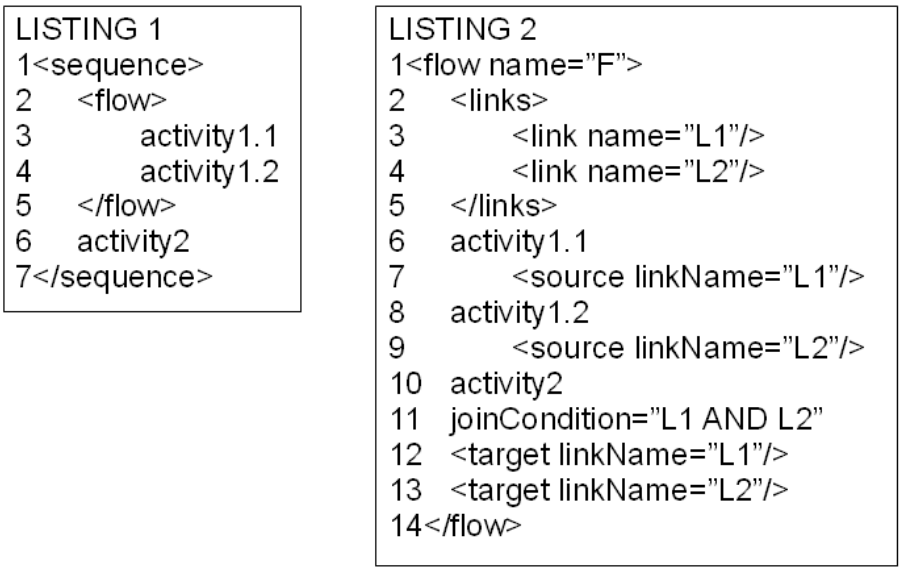

Figure 2.9: XLANG and WSFL-styles (Adapted from [13])

The listing examples in Figure 2.9 give us a perspective through the two different impacts XLANG, and WSFL provided in BPEL. Listing 1 is in XLANG-style, where routing is

through structured activities, while Listing 2 illustrates the link usage, reflecting the WSFL-style [13]. Just as the example shows, each element in the process is called an activity, and these activities are either basic (primitive), or structured. Structured activities help sequencing the activities, while basic activities help explain what happens during the process flow within a specific activity.

Basic activities can be listed as <invoke> for invoking an operation of a Web service described in WSDL, <receive> for receiving, <reply> for replying to the operations that the process itself exposes, <fault> for throwing faults, <wait> for waiting, further activities, you may refer to Table 2.1.

Table 2.1: Primitive Activities

Basic activity name	**Activity**
<invoke>	Invoke an operation of one of the Web services described in WSDL
<receive>	Wait for a message from an external source
<reply>	Reply to an external source
<wait>	Wait, remain idle for a while
<assign>	Copy data from one variable to another
<throw>	Throw execution errors
<fault>	Throw faults
<terminate>	Terminate the whole service instance
<empty>	Do nothing

Table 2.2: Structured Activities

Structured activity name	**Activity**
<sequence>	Define an execution order
<scope>	Group activities to be treated by the same Fault handlers, Event handlers, Compensation handlers, And scoped variable definitions
<flow>	Parallel routing
<while>	Loop
<switch>	Conditional loop
<pick>	Non deterministic choice

Table 2.3: Additional usages in BPEL

Tag	**Explanation**
<partnerLink>	Execution order can further be controlled through them Useful to define dependencies between activities Can be one, or two-sided*
<variable>	Store messages that are exchanged between partners Hold data about the state of the process

*One sided partner link states that either partner or the process act as a pure client of the other. Two sided partnerLink however is the one where process is invoked by the partner, and also invokes the partner's service. On the other hand, structured activities are consisted of other activities. Those activities are nested within structured activities, and structured activities impose control over them, and provide them with common properties. Structured activities can be listed as <sequence> for strict sequencing, <while> for looping, <flow> for parallel routing, <pick> for non-deterministic choices, <switch> for conditional routing, <scope> for grouping activities to be treated by the same fault handlers, and for further information, we may refer to Table 2.2. Besides the activities defined above, BPEL language provides the specification of relations among Web services in the business process via <partnerLink>. Additionally, BPEL allows us to declare some variables by using <variable>, shown in Table 2.3 [20].

CHAPTER 3

PROPOSED APPROACH: Service Oriented Architecture with Axiomatic Design

Decomposition has been the target concern for designers, such that different methodologies have been proposed to comply with the decomposition problem. When the designer is concerned about the implementation process of the system, the decomposition ends up with being a structural decomposition [15]. On the other hand, if the designers consider the system as a sum of business processes, and how to implement the processes is not a concern of these designers, since their company may hire some other companies to realize the system, or designers use remote services, then the consideration is in process level, and is called process decomposition [16]. Each and every decomposition approach cares about the reusability issue, where the decomposition of the system should be achieved with respect to the existence of the corresponding implementation units. Additionally, the implementation units in the decomposition have been subject to change. The decomposition concerns started in the history with the function emphasis, and then shifted towards objects with the introduction of object-orientation in the software world, and then it moved on to components, and lastly towards services. The software environment has recently been introduced with business processes, where each component in the decomposition is a well defined collection of structured and related activities, where a specific service for a specific customer domain is satisfied. In our methodology, both business processes, and services are used as units of decomposition.

A Service Oriented Architecture (SOA) approach has been introduced with the process decomposition in [12], and what we propose in this research is not only a contribution through the decomposition process, but also a supporting mechanism for the process integration phase. In this research, we have extended ADCO [14] with SOA, and introduced a new methodology, named as Axiomatic Design in Service Orientation (ADSO). This methodology uses

business processes and Web services for different levels of process decomposition. This new methodology considers the Axiomatic Design Theory (ADT) approach to be used in the development of service-oriented architecture compliant systems. Let us introduce the axiomatic design part of the research, and then we shall move onto ADT introduction with the suggested service orientation.

In order to comply with the demand and reach a solution, a top-down approach has been followed, where the big problem has been divided into finer demands. This decomposition helps the designers to make customer requirements more understandable, and see how much they can fulfill these requirements with the existing physical solutions. In our methodology, to fulfill such a top-down approach, ADT has been chosen, and the procedure for ADT is as follows. Firstly, the system engineers, system designers and the customer meet, and they agree on the Customer Needs (CN). Then these CNs have to be converted into Functional Requirements (FR), so the CNs are evaluated in the designers' and developers' perspectives, and corresponding FRs are listed to be resolved for the system solution. Designers represent the FRs, in other words the system requirements, in a hierarchy. They are required to cover all the customer expectations. On the other hand, how these expectations are achieved is the concern of Design Parameters (DP). The DPs are the physical structures, such as components, services, methods, or Web service interfaces. In ADSO, Web service interfaces correspond to DPs in ADT. While DPs give out a characterization of the design, the actual implementation is achieved in Process Variables (PV). In ADSO, the so-called zigzagging process is conducted among function, physical, and process domains as discussed in the ADT definition. The designers shall decompose the domains while zigzagging bearing in mind the conditions about the FRs, and existing DPs. If any FRs are existing in hand, but no corresponding DPs are available, then corresponding DPs, and their actual Web services are to be implemented. Vice versa, if we are utilizing a better established and engineered domain, we shall decompose our FRs into finer FRs in order to make use of our existing services. Following the ADT methodology steps, when decomposition and mapping activities are complete, a Design Matrix (DM) is constructed, where we can keep track of which DP satisfies which FR. The developers should try to make sure that the FRs are satisfied by independent DPs, in other words, each DP satisfies only one FR, if possible. In case multiple DPs may satisfy a particular function, determination of which DP to use can be reached based on the information axiom of ADT. With the help of independence and information axioms, we identify the design as ap-

propriate and the ultimate one, or in need of more decomposition. As stated in the definition of the independence axiom, the FRs are supposed to be independent from each other. This is the idea behind the best design, so that when the system is to be modified, and FRs are subject to change: they can be removed, replaced, or extended without interfering with the other FRs, and their corresponding DPs. On the other hand, the information axiom states that the design should have the possibly minimum information content, and this is mathematically evaluated through the equation given in Equation 2.1. If the independence axiom is achieved within more than one design matrices, then the information axiom is used to decide which design matrix proposes a better design. Since we have decomposed the big problem into manageable pieces, and come up with corresponding independent solutions, we have actually concluded the first step for the top-down approach in ADSO.

Our next step in ADSO methodology is to conclude the top-down approach with a graphical representation. The SOSE modeling language [12] is a graphical modeling language that supports the top down decomposition approach in complex business processes. However, our approach supports the SOSEML hierarchical representation tree with a design matrix. Our SOSEML hierarchical representation tree also provides us with a supporting mechanism, where each process node in the tree can be explained with an activity flow diagram by using a plugged-in BPEL designer. This activity flow diagram construction job is actually a bottom-up process support, which will lead us to build the pieces together, and construct the big system as the final product.

SOSEML defines the system as the root process, and then this process is decomposed into sub processes to reach the atomic processes. In SOSEML, the atomic processes, which cannot be decomposed into any more sub processes, are the Web services. Web services declare methods to serve for the received requests [1], and a web service can have more than one Web service interface, where each interface can declare more than one method. The process, Web service, and interface figures in SOSEML tree are presented in Figure 3.1.

In order to make the Web services work together, coordination of Web services is required, and for the purpose, WSO, and WS-Choreography are used. These coordination mechanisms are used in the root, and sub processes to specify and order the leaf level processes.

The procedure to apply ADT to service orientation is the last step to explain in our top-down proposition. As indicated in the ADT definition, the DPs satisfying an FR may be constructed

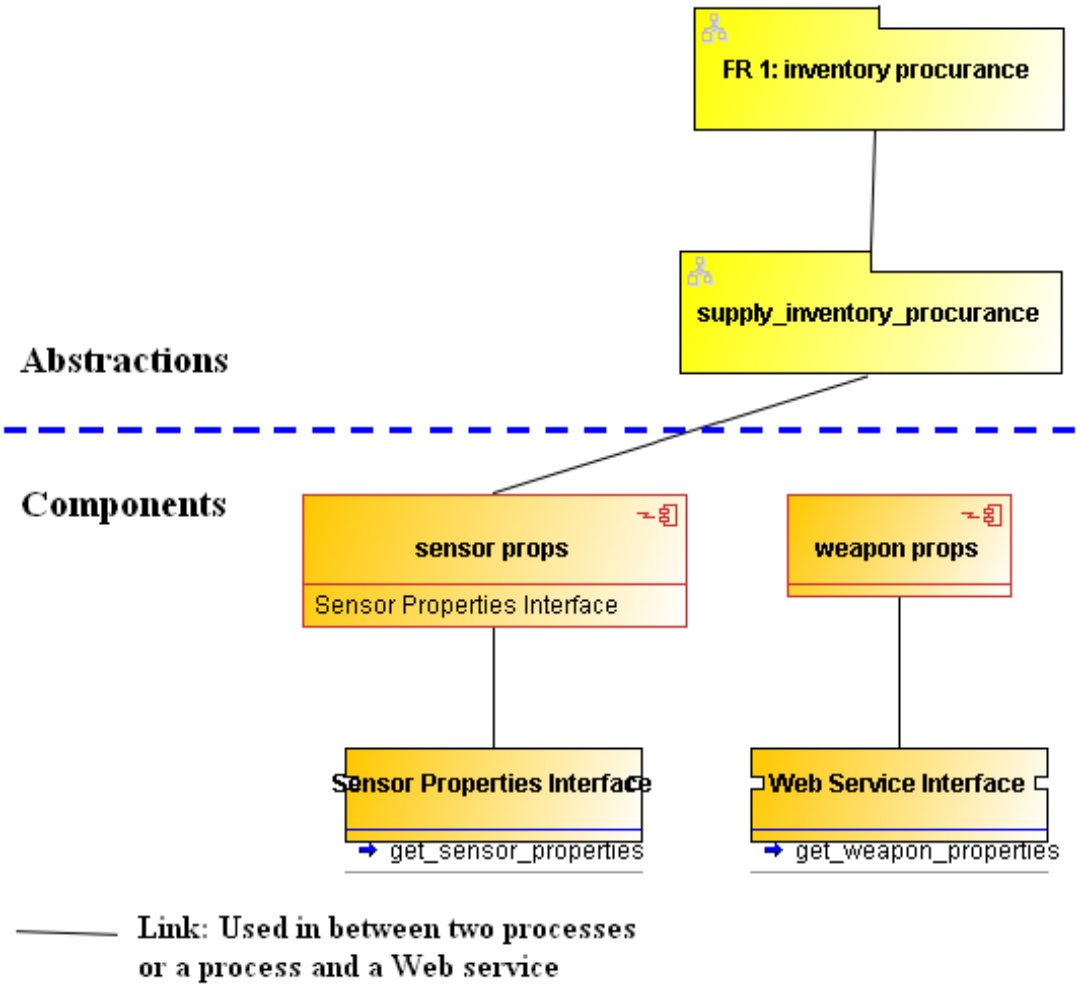

Figure 3.1: Process, Web service, Web service interface, and link figure samples

via using various Web services. At this point, the DPs are used more than once to satisfy various FRs. Although satisfying an uncoupled or decoupled design matrix is meant to develop by the end of the top-down approach, this spoken situation is something we will often encounter in real life problems. At this point, the SOSEML tree we create is formed using every aspect of ADT's design matrix, which contains FRs, DPs, and their relationships. The algorithm we have designed to implement such design matrices is as follows; we had to consider all functional requirements, and their corresponding design parameters. Considering the relationships among them, we have built sub processes containing relational process variables, which correspond to existing Web service implementations, and their corresponding interfaces. The root node is visualizing the whole process that would realize the ultimate business goal. Then the functional requirements are abstracted to a comfortable level, where sum of

all functional requirements correspond to the whole system. These abstracted functional requirements are linked to the root node as sub processes, and the Web services are linked to the related functional requirement sub processes with the help of the relational design matrix we have in hand.

As in the SOSEML tree approach, the root node corresponds to the process node for the whole application. Then the root process is decomposed into sub processes corresponding to the functional requirements in the design matrix we have built. Each functional requirement is solved by using multiple design parameters. These design parameters correspond to different Web service interfaces, which are defined during Web Service Design, yet the sub processes corresponding to the functional requirements of the system are decomposed into the Web services that are able to cover their requirements. With this approach, we will contribute to the development period in such a way that the SOSEML tree will give the designers a clue about the process flow that has to be achieved in order to reach the ultimate system design. The Web services below the sub processes are able to be choreographed in their corresponding sub processes, since the design matrix reflects all the needs of functional requirements in its rows. The SOSEML tree is composed of processes, where the leaf level processes use the existing Web services, thus the processes at the first level are the orchestration of Web services. The first-level processes include the Web service interactions, variable assignments, input parameters, and operations which are defined in Web service WSDL files. In SOSEML tree, the rest of the process nodes; the root and intermediate processes are composed of sub processes, so the process models of these processes are the choreography of the sub processes. Since the orchestration of existing Web services done for the leaf level processes compose composite Web services, the intermediate and root processes actually still orchestrate the Web services, which are in this case, composite Web services.

The last step of ADSO methodology proposes to support the process integration phase. Although this stage may not be viewed as having full process integration support, the idea behind this stage is to help the integration of the activities within each process on the SOSEML hierarchical representation tree. The activity flow in each process can be modeled by using a business process designer. In our case, we have decided to use the BPEL Designer, where we have delivered the process model to the developer by considering only the Web services that are connected to that particular process on the SOSEML hierarchical representation tree. Internal details of each process can be modeled by the developer with the BPEL Designer

in ADSO, which is an open source BPEL designer interface provided as an Eclipse plug-in. Service oriented software engineering recommends the business process modeling phase to start from the leaf level of SOSEML hierarchical representation tree, thus follow a bottom-up approach. The reason for leaf level start up is because the web services are composed to form composite web services, and the newly formed web services can serve for the modeling of the above level business processes. This process would then end up with the choreography of the sub level processes, and combine to resemble the system solution.

What is accomplished by the approach is the introduction of a methodology for building software systems out of Web services. The support from the ADT is utilized, for simultaneous decomposition of requirements and design, allowing early modifications to both workspaces since the "zigzagging" suggests to observe the effect of every individual decomposition activity in the other workspace (domain, in ADT terms). Also supported with the Information axiom to select the better fitting design among alternatives, and the analyses that can be conducted on the design matrix for less coupled designs, this approach comes with additional advantages. The steps corresponding to the development starting from the requirements analysis can be summarized as:

1. Decompose requirements
2. See if existing design parameters match the current requirements: decompose design if not:
   (a) If design suggests a different organization of requirements, goto step 1 and reconsider previous decomposition of the requirements
   (b) If all requirements items are met by the design parameters, both requirements and design are finalized and they are in agreement. Else goto step 1 for decomposing the next requirements item.
3. Check the design matrix for coupling- modify for less coupled designs if possible.
4. See if alternative designs can be feasible repeating steps 1 to 3.
5. If there are alternative designs, evaluate them using the Information axiom for the best fitting set of Web services that have minimal extra functionality.
6. Treat Design parameters as BPEL process models:

7. If a process model is a leaf-level (atomic) process, model it as a Web service - it is no longer a BPEL process.
8. Try to execute the BPEL processes, starting from the bottom ones - to refine and finalize the BPEL representation
9. Go upwards on the tree, for different BPEL nodes until the root is reached.
10. Composition is finished.

The methodology implies a top-down decomposition along with the problem definition and supports this activity with matching solutions. Then, the system is composed in a bottom-up manner. The core of the methodology is thus presented; however, this only represents the two ADT domains that are the FR and the DP representing requirements and design. A further reaching support in terms of the lifecycle can be achieved by repeating the zigzagging for the Customer Needs and the FR domains, and the DP and the PV domains. In the later case, basically web services will be decomposed to methods - that is especially valid if web service development is also inevitable.

CHAPTER 4

A CASE STUDY: DESIGNING AND MODELING A MILITARY DEPLOYMENT PLANNING SYSTEM

In this chapter, we will be explaining our methodology with a real life example. First, we will introduce you with the domain description, where we will provide information on military deployment planning system, then we will move on to explanations on software Analysis, system Design and modeling phases.

4.1 Domain Description

In this last chapter, a SOA based software system is modeled with SOSEML to demonstrate the basics of ADSO methodology. This case study is developed to analyze the system, propose a design alternative to model the system, create a hierarchy of system's process models, and practice the BPEL designer via a process flow study on some of the processes in SOSEML hierarchy view. Web services may be composed on the Web Service Design part of the ADSO tool. In this case though, they are just supposed to be existing in the solution domain, and not actually implemented. We have also supplied the designer with the possibility of using abstract Web services, and they can be defined in the Application Design part of the ADSO tool. A SOA based methodology has been introduced by Eren Koçak Akbıyık in his thesis study, using existing, but not implemented as are in our case study, Web services. Our studies have shown that the system designed in Akbıyık's work has been constructed with a top-down approach, but the design technique which would support this decomposition was missing. In other words, there was a gap in between customer needs and SOSEML representation. Since ADSO methodology proposes an approach starting from taking the customer needs until process flow representation in BPEL designer, our ADSO methodology is an appropriate match

to cover this gap. This case study has been taken from Akbıyık's study, and re-conducted, to show the whole process flow from taking customer needs until BPEL representation of processes in SOSEML hierarchy view. In order to give a complete system management perspective, and get a whole picture of the flow, we have conducted a reverse engineering on the previous study. Then a detailed coverage plan has been made on the Web service interfaces that Akbıyık has introduced, and the functional requirements have been obtained. After obtaining the functional requirements of the system, since we have obtained the necessary components for applying ADT to the system, we have started to go through ADSO steps. In the first part, we have applied axiomatic design methodology on the functional requirements, and design parameters supplied, and obtained an FR-DP design matrix. Then we have followed the hierarchy of design parameters that has been formed on the design matrix, and used this information to implement a graphical representation using SOSEML. Each process is identified with a corresponding shape, but the flow within a process had to be shown via an extra designer. At that point, BPEL designer helped us to introduce the process flow.

Let me first introduce you with the system concepts of this case study, [12]. The purpose of the system is to decide on a deployment plan for the deployment of a number of weapons and sensors in the air defense operations of the military critical regions. There are two types of units to perform the air defense activities for a geographical region. These are the weapons, which have a target line for facing its firing target; the sensors are radar units to track the air. They are placed at appropriate locations in the zone, and send the tracking information of tracked air vehicles to the management center for evaluation. The gathered tracking information in the management center is used to identify the air vehicles, and classify them as friend, or foe. According to the identification and classification results, appropriate weapons are engaged to the foe targets, and are fired when the field manager gives the order. A deployment plan is consisted of the placement information, and task assignments for the weapons and sensor units. There is also battlefield geometries used to indicate specialized defense areas within the defense zone. The battlefield geometry placement information is also included in a deployment plan. Unit placement is a crucial job. Weapons should be placed in the correct defense positions considering their capabilities, such as target prevention and range capabilities. Placements of the units in the geographical terrain are crucial. On the other hand, sensor units should be placed in suitable locations considering their radiation and coverage properties. The correct placement for sensor units may require some analysis accomplished by some

geographical information systems support unit. The units, their task assignments, and some detailed information is given in Table 4.1.

Table 4.1: Units and their corresponding task assignments.

UNIT	TASK ASSIGNMENT	DETAIL
Weapon	Primary target line (PTL)	PTL indicates direction information that weapon is targeted to.
Sensor	Radiation segment (SRS)	SRS indicates sector regions sensor is responsible for tracking.
Battlefield Geometry (BFG)	Three dimensional visual geometry; circle, polygon, corridor, line	Identify critical areas in defense zone

In addition to the abstract information given in Table 4.1, according to the working properties of a sensor, it may track a full circular region, which is a 360 degree horizontal coverage or several separate sectors. On the other hand, BFGs are defined for both air and land area geometry visualization to identify the hostile tracks. The prohibited areas, minefields, restricted zones, and airfields are marked with BFGs and in case of intrusion through these areas, the management center evaluates this as a hostile intrusion, since air and land forces are aware of the prohibited areas, and would not violate the areas. The BFGs are valid for certain amount of time, and updated by air and land forces on a regular basis.

In the next section, ADSO methodology is described on military deployment planning project. The ADSO methodology steps are applied to the given customer needs, and SOSE model is depicted for each corresponding Web service, and system processes.

4.2 Software Analysis, System Design and Modeling

In this section, we will introduce the military deployment planning software, and the reverse engineering phase we have conducted. Then we will provide you with the two designing phases, which are Web service design, and the application design.

4.2.1 Introduction to Military Deployment Planning Software and Reverse Engineering

The military defense deployment planning software takes the defense region, the weapon, and sensor inventory, and prohibited area information as the input, and introduces a defense plan output for the given defense region. The defense plan consists of placing the weapons, and sensors, and assigning their tasks, or we shall say orders in military terminology for the units. For more details on military deployment planning software, one should refer to [12].

The ADSO methodology proposes two main steps to evaluate the system. First, develop a relational diagram consisting of the FRs, and DPs, in other words, the FR-DP design matrix, then decompose the system into business processes using SOSE modeling technique via using the relational diagram composed in the first step. In addition to these two steps, we can define the process flows within each business process using BPEL designer. We shall go into detail for modeling of the military deployment planning software now.

The military deployment planning software is capable of accessing the services supplied by multiple army forces, as well as commercial ones. In this study, the Web services are not actually existing services, but are supposed to be existing in the solution domain. As we have stated in previous chapter X, the Web service names, and occupations are described in Akbıyık's case study, but corresponding functional requirements are not considered. First, we have considered to what kind of functional requirements the customer domain could have been mapped; before stepping into the solution domain. Here is a list of functional requirements delivered from a reverse engineering study on the Web services listed in [12]. We should note that the X indication for a weapon or sensor throughout the list, corresponds to either a sensor or a weapon indicated with its identification number to be used in the corresponding service. An X for an army corps is likewise, an army corps indicated with its identification number.

Possible Functional Requirements List

- Get weapon identification numbers
- Get sensor identification numbers
- Get all army corps inventory identification numbers
- Get army corps X's inventory identification numbers

- Get army corps X's weapon identification numbers
- Get army corps X's sensor identification numbers
- Get weapon type X's properties
- Get sensor type X's properties
- Set weapon X's coordinates
- Set defense region
- Set defense point
- Set sensor X in appropriate position
- Assign primary target line of weapon X
  - Analyze defense region
  - Analyze range properties of weapon X
- Combine analysis results of
- Analyze defense region
- Analyze range properties of weapon X
- Analyze visibility of sensor X
- Analyze coverage of sensor X
- Analyze sensor working properties for selected defense region
- Get sensor radiation segment for sensor X
- Get air area battle field geometries from the air forces
- Get land area battle field geometries from the land forces
- Get vector sheet for the defense region
- Get raster sheet for the defense region
- Get relief sheet for the defense region

- Prepare map layer presentation
- Prepare information layer presentation
- Get information and map layers presentation on a GIS panel
- Get separate layer presentations on a GIS panel

Since we have defined possible functional requirements for the proposed system, we shall map the functional requirements with our existing Web service interfaces. From now on, we will be demonstrating our following steps in our ADSO tool. We will be demonstrating some of the Web service designs with their design matrices, and SOSEML representations, and then we will conclude with the application design matrix and application SOSEML hierarchy tree. A couple of BPEL graphical representations will also be given, yet the whole BPEL graphical representations can be reached in [12].

4.2.2 Web Service Design

Throughout this case study, since the Web services are assumed to be existing ones, the system design has been done with Web service interfaces, that were created on-the-fly. In real world, the designer should have a list of available web service interfaces that belongs to a specific domain, and their relevant Web services should be ready to use. The application we are working on, or the actual applications the designers will be applying the ADSO methodology shall be contained within the previously studied domain. We have chosen *Get all army corps inventory identification numbers* functional requirement, and chosen the corresponding Web service interface we have maintained during our reverse engineering study. A design matrix as in Figure 4.1 has been constructed.

The *get_inventory_ofAllArmyCorps* Web service interface is thought to be supplied by multiple army corps Web services. When the relevant *get_armyCorps_inventory* Web service of an army corps is invoked, an array of inventory identification numbers are thought to be gotten. Once the design matrix in Figure 4.1 is saved, the SOSEML hierarchy tree of the Web service can be required from the ADSO tool, and Figure 4.2 is provided.

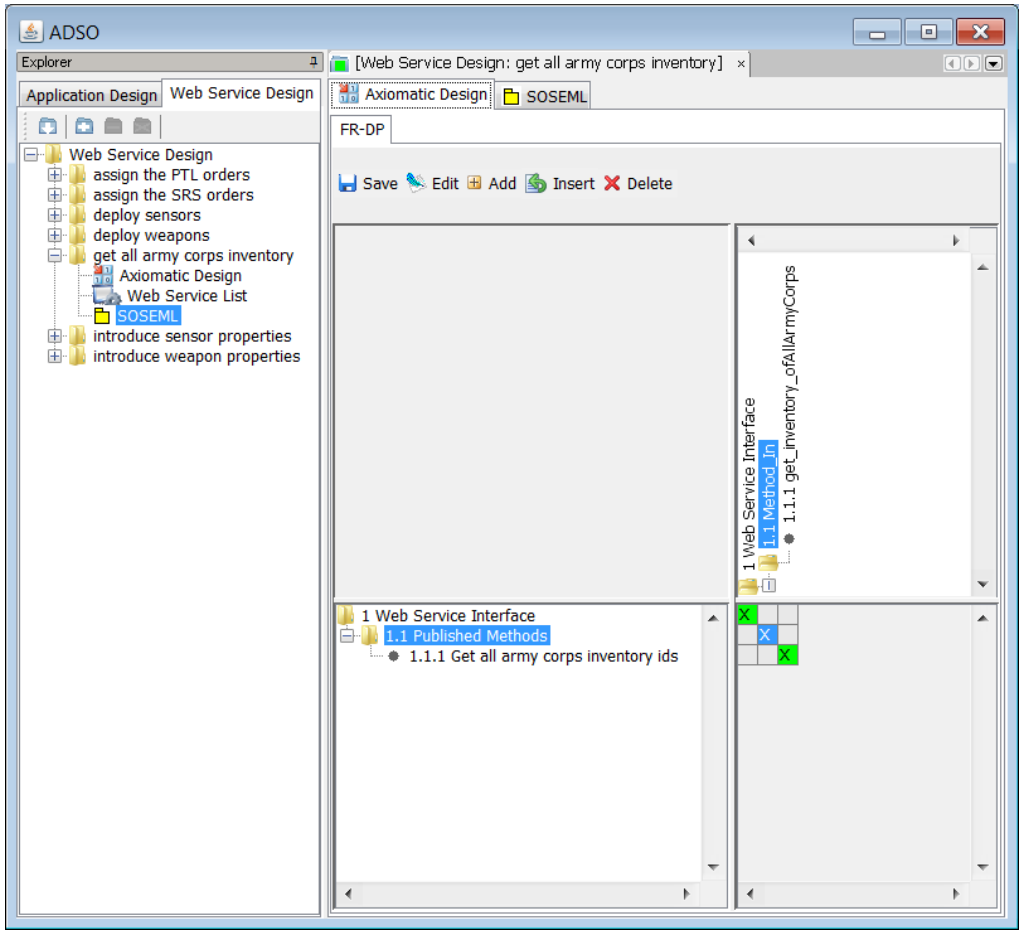

Figure 4.1: Design matrix for *Get all army corps inventory* Web service

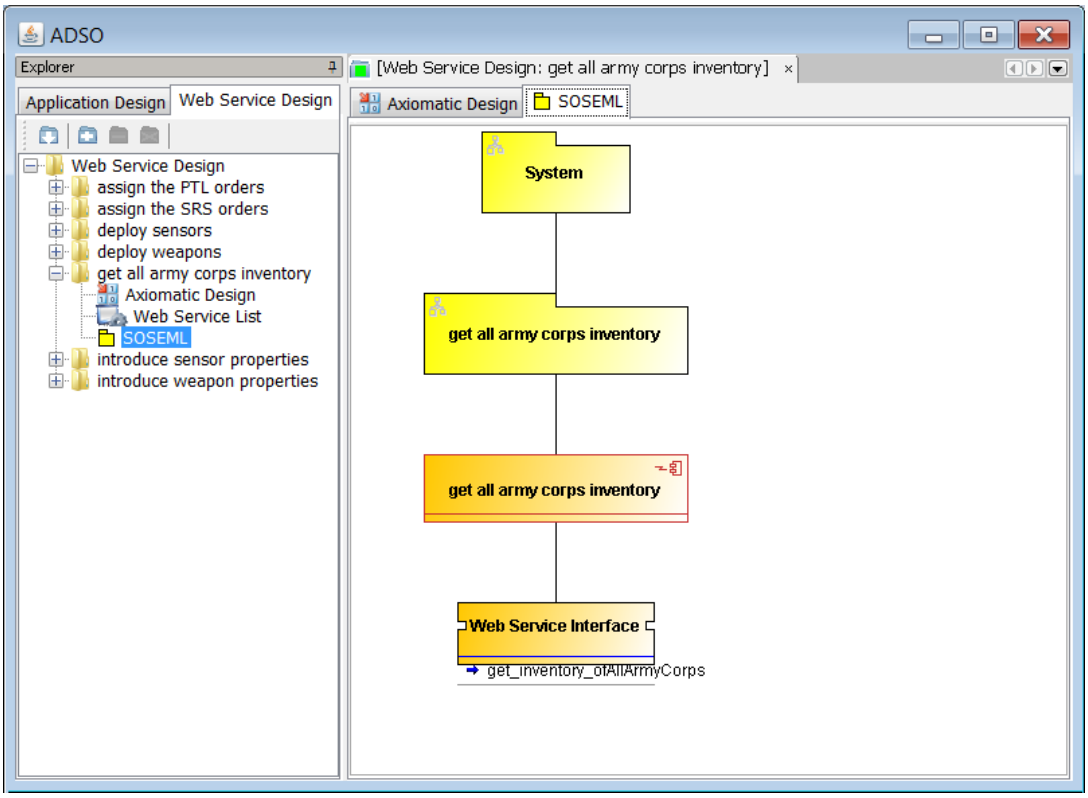

Figure 4.2: SOSEML representation for *Get all army corps inventory* Web service

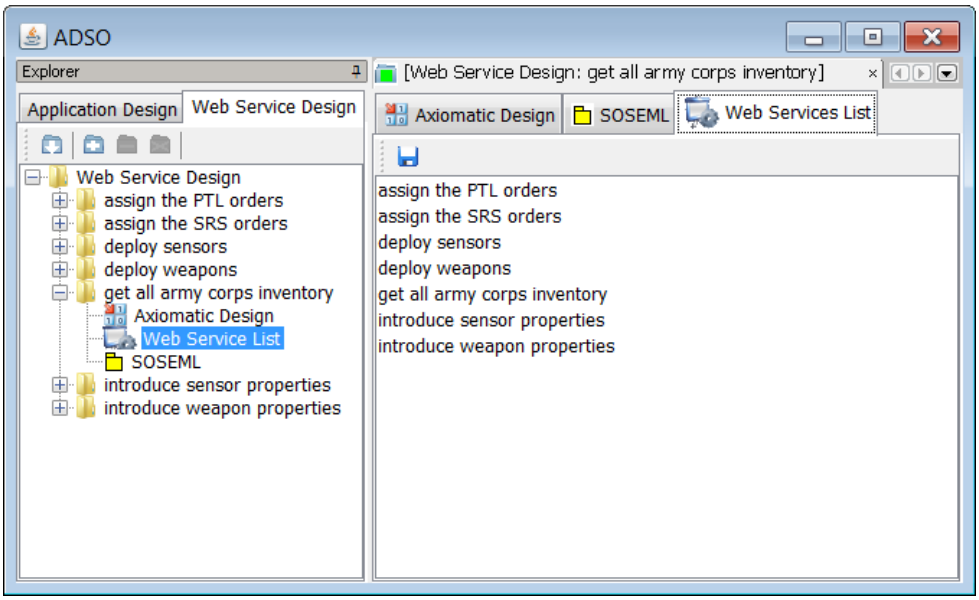

Figure 4.3: Web service list provided at any stage of designing study

While the designer works on her newly introduced Web service creations in Web service design section, or on the application design, the Web Service List will be providing all the available Web services that are ready-to-use for the designer. Figure 4.3 provides us with the current Web service list.

After these stages are complete, the designer may construct the *get all army corps inventory* process's flow by double clicking on the process figure. As shown in Figure 4.2, the icon on top left of the process figure is off. Once the BPEL graphical representation is constructed, the icon will be lit.

4.2.3 Application Design

In this section, we will provide you with the ultimate design matrix of the *Military Deployment Planning System* software application, and its SOSEML tree representation. The functional requirements provided in section 4.2.1 are used in the development of Military Deployment Planning System's design matrix as depicted in Figure 4.4.

After the basic relationships are inserted into the design matrix, we have also marked the input output relationships on the matrix, which as stated in section 3.2, contributes to the future integration phases. We have saved the designed matrix via pressing on the *Save* button, and double-clicked the *SOSEML* list item of the corresponding *Military Deployment Planning* application item on the *Application Design* list. Finally, we meet with our Military Deployment Planning Software SOSEML hierarchy tree as depicted with the screen shots starting from Figure 4.5, and following with Figures 4.6, 4.7, and 4.8.

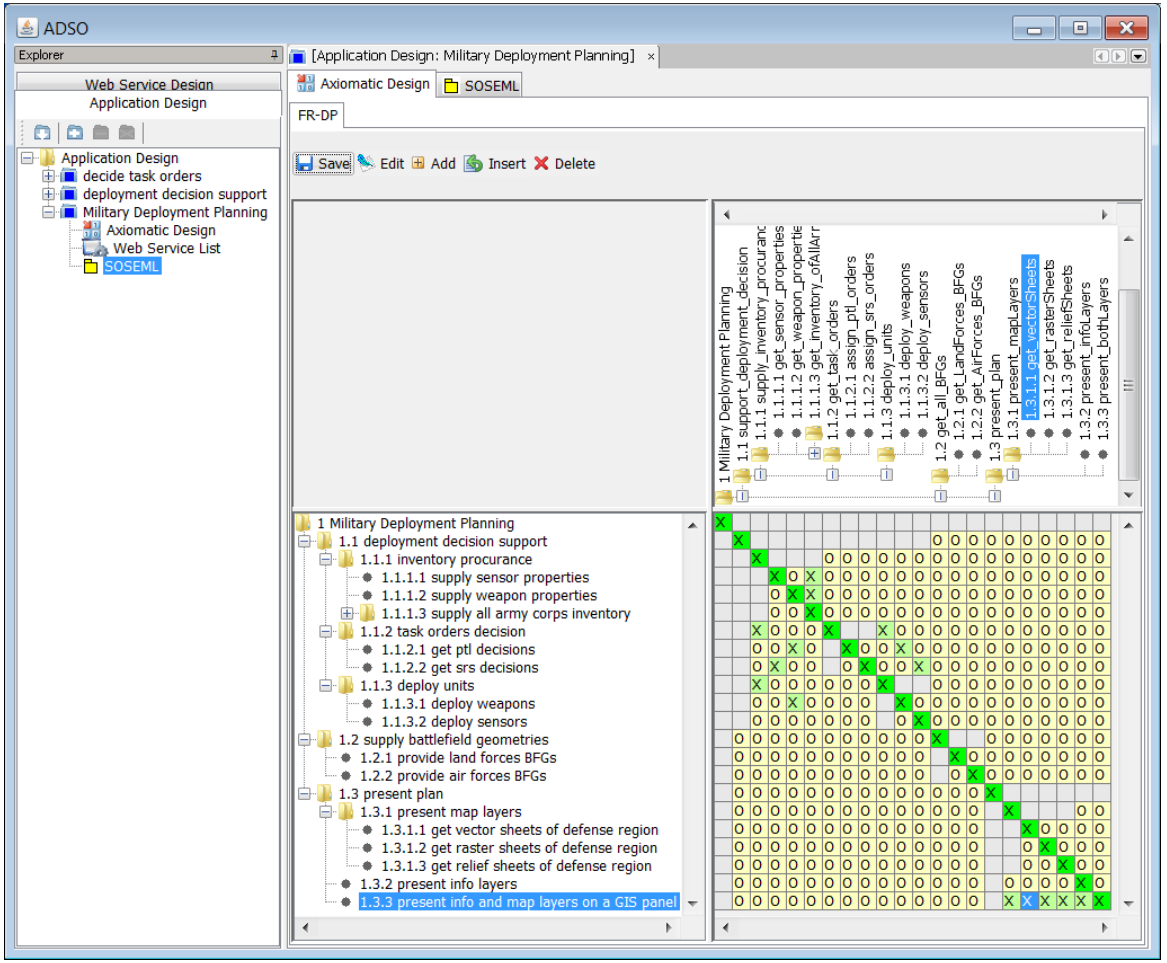

Figure 4.4: Application Design Matrix for Military Deployment Planning Software

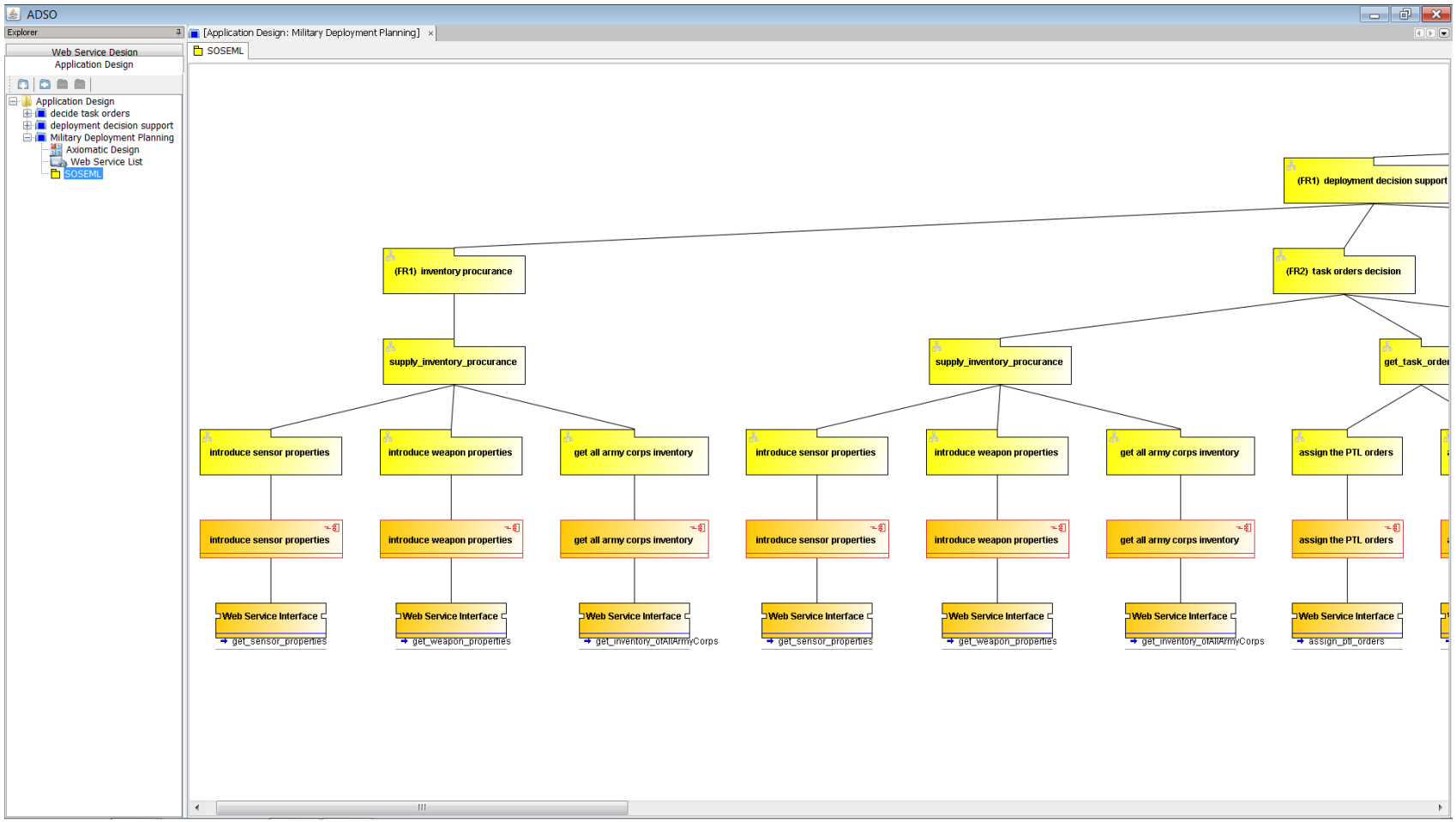

Figure 4.5: Application SOSEML hierarchy tree for Military Deployment Planning Software

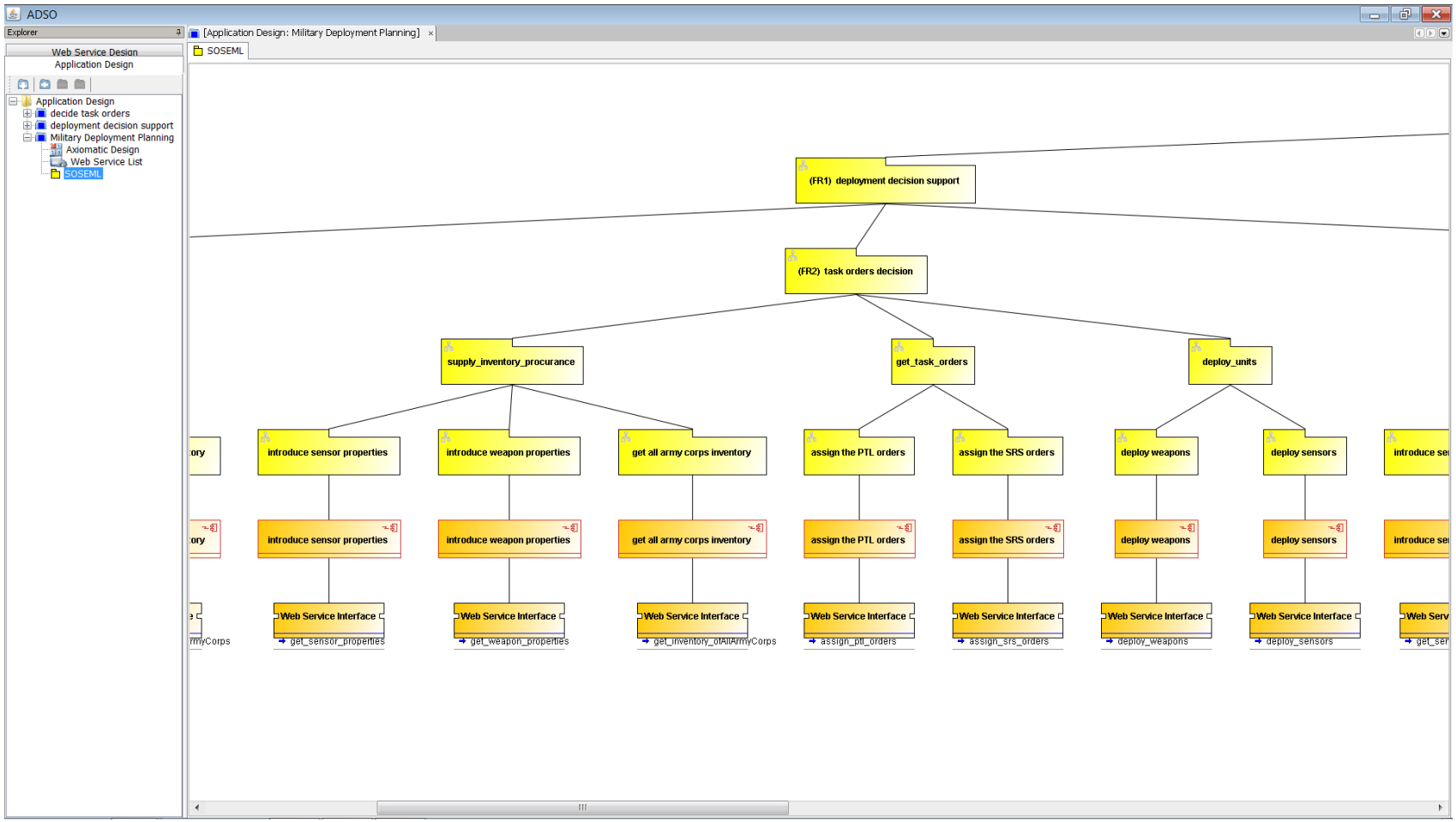

Figure 4.6: Application SOSEML hierarchy tree for Military Deployment Planning Software (continued)

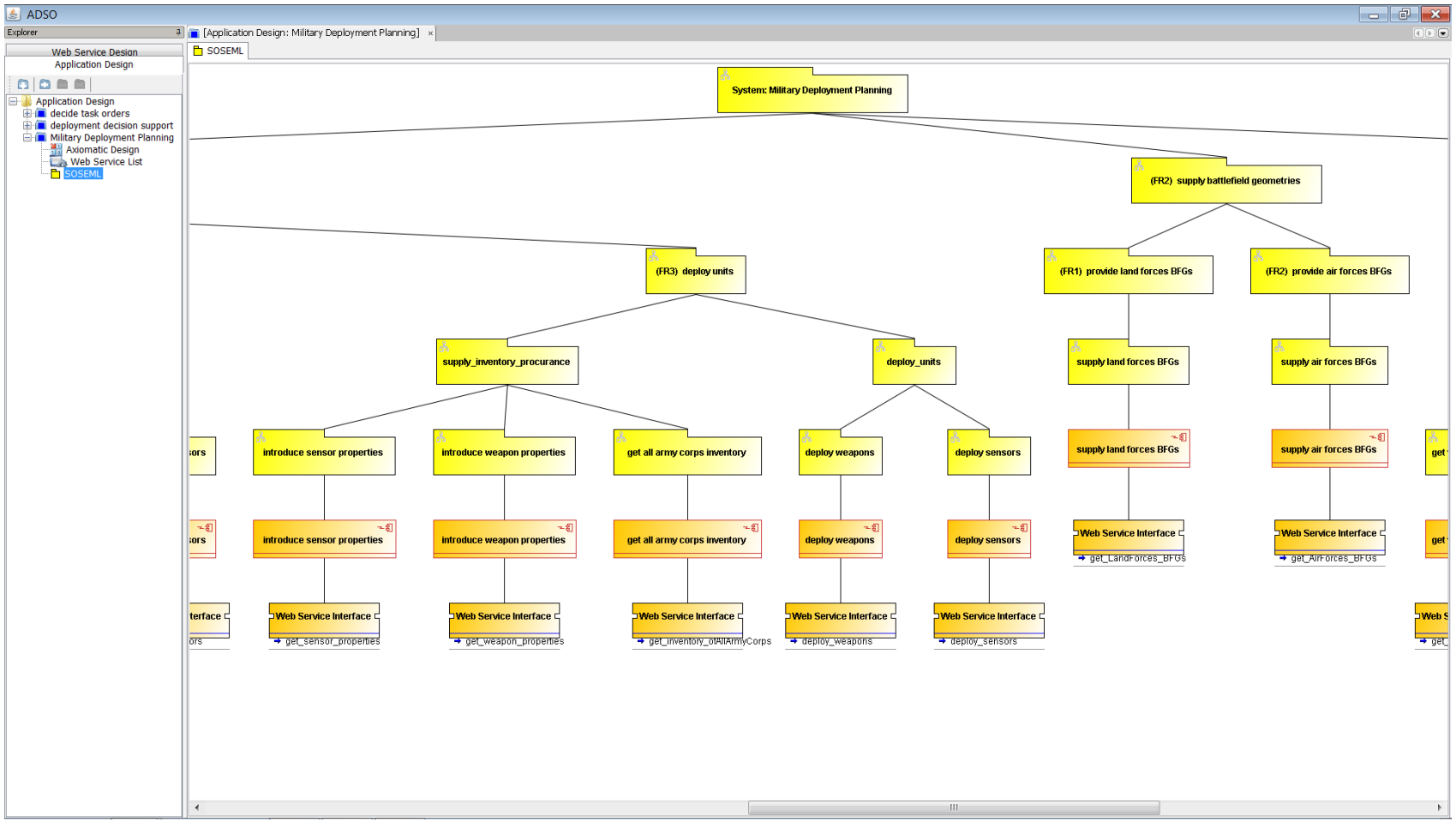

Figure 4.7: Application SOSEML hierarchy tree for Military Deployment Planning Software (continued)

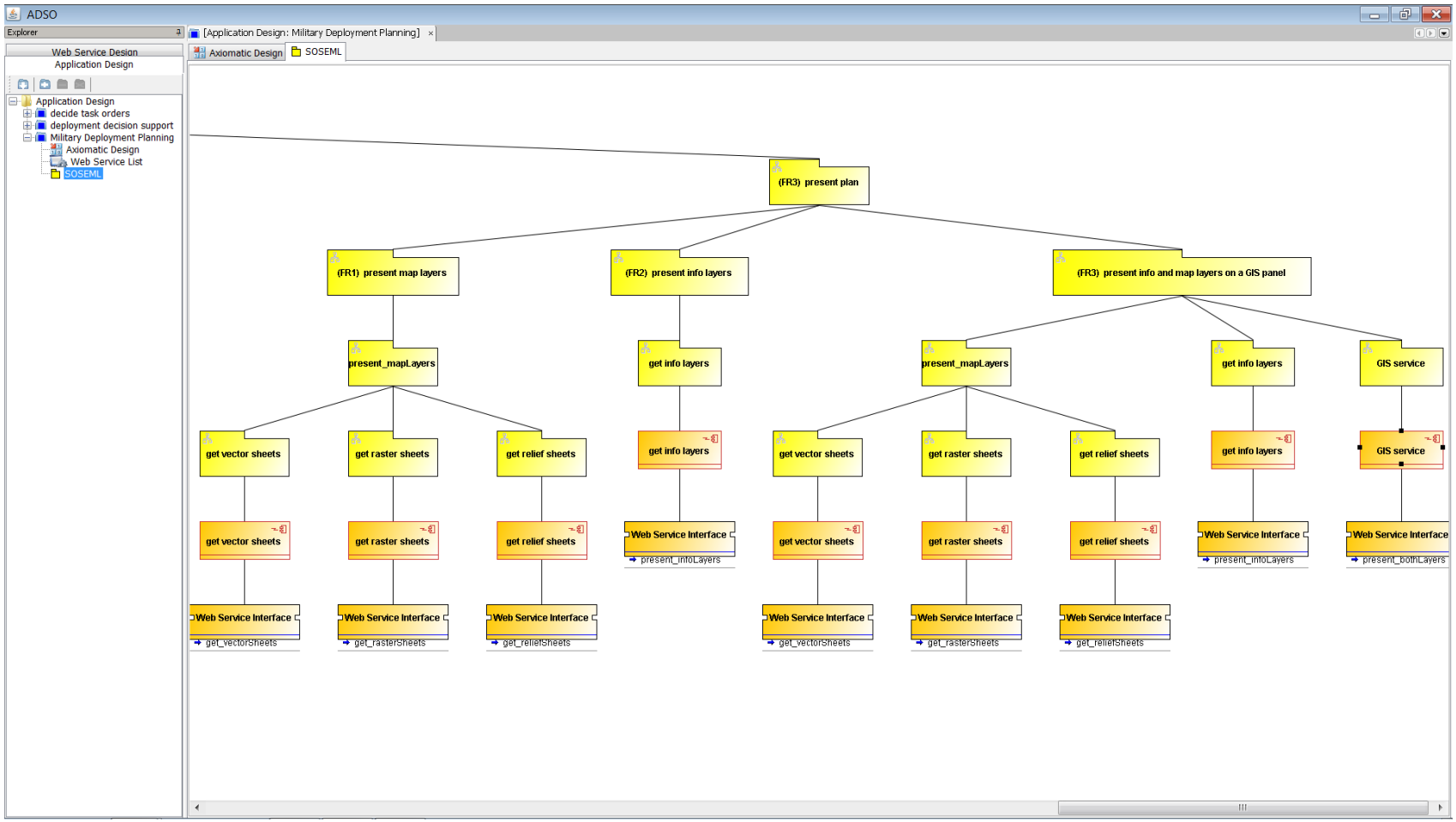

Figure 4.8: Application SOSEML hierarchy tree for Military Deployment Planning Software (continued)

CHAPTER 5

CONCLUSION AND FUTURE WORK

In ADSO methodology, we have introduced the two concepts ADT, and service-orientation, and proposed them to work in coherence. A previous study has been conducted for ADT, and component-orientation, but since the approach towards the usage of web services has become more popular in the software market, the conjunctions of ADT, and service-orientation has attracted our attention, and we believe that it would find the place it deserves, as more systems are realized by using this mechanism. What we have intended in this research was not only to provide a contribution to the decomposition process, but also to supply a support mechanism for the process integration. To be more specific, in ADSO methodology, we have proposed to satisfy the reusability issue by using the existing Web services effectively, during the system decomposition. We have realized this by using axiomatic design approach, where functional requirements are mapped with the design parameters, and zigzagged among the functional domain, and physical domain. A simultaneous decomposition in functional domain, physical domain, and process domain provides the designer to decompose one domain while considering the solution domain. This supports the designer with the ease of handling customer needs with existing web services. Our second purpose has been to take integration phase into consideration during decomposition phase. By this way, we would have supplied a smooth transition between the decomposition and integration phases. We have realized our methodology with the help of ADSO tool that we have constructed. We have considered the multiple physical variables to realize each and every functional requirement in the design matrix, and constructed the SOSEML hierarchy tree with the relevant process domain components, which are the Web services in our ADSO methodology. The initial ADT approach for OO development had to consider the methods as the implementation units in the ‘functional domain’. By replacing these units with web service interfaces, reusability has been achieved. Rather than being a prescriptive approach to development, targeting code writing, with our contribution

ADT can now support compositional development.

ADSO has a product line support understanding starting from the definition of the FRs of the system, mapping them to the existing, or abstract Web Service interfaces, building the FR-DP design matrix, graphically introducing the Web service orchestration, and choreography, and letting the developer model an activity flow for each process in SOSEML representation of process model hierarchy, via using BPEL designer. Although the promised product line supporting mechanism has been achieved in ADSO, the ability to add Web service interfaces on the fly on SOSEML model is also implemented. In order to supply further support, the reverse line can be tailored, where Web service additions made on SOSEML model are reflected to the domain design via implementing the corresponding Web services, and reconstructing the design matrix by including the newly introduced Web service interfaces, and their possible functional requirement fulfillments. Thus, these interfaces would then extend the functional requirements, supporting a broader customer domain.

Another future work may be adding feature models to the ADSO concept, where the customer would feel comfortable with the needs she would be providing to the designer, since she would have a chance to see feature models for similar domains to predict her needs in her application.

Another future work may be building an orchestration engine, where the grammar of BPEL designer can be executed. This orchestration engine would not only coordinate the activities, but also compensate the overall process when an error occurs. This may be tailored to current ADSO tool as a last step definition. Another future work could be an automatic activity flow engine, where the process flow would be predicted from the SOSEML representation of process model, and composed automatically in BPEL designer. In ADSO methodology the indicated future works have been left as a future work, since our ultimate goal was to define a supporting tool for the product lines starting from stating the functional requirements until designing the activity flow of each process in the SOSEML representation of process model hierarchy. The ADSO methodology we have provided has filled a gap in software environment. There exist various decomposition approaches in the software world, but with the help of ADSO, service oriented decomposition has reached a supporting mechanism that handles the design process starting from taking the customer needs, and decomposing corresponding functional requirements, and design parameters within a structural visualization, which is the design matrix. We could have decomposed the function, physical, and design domains con-

currently, but we would not be able to show them in a well understood structure unless we had the user-friendly methodology, the ADSO. We could have started from an earlier phase like providing the customer with probable customer needs we have collected from existing domains, and give them a feeling for how to approach their problem, and provide the designers with the customer needs thoroughly.

REFERENCES

[1] Akbıyık, E.K., Süloğlu, S.,Togay, C., and Doğru, A.H., *Service Oriented System Design Through Process Decomposition*, Proceedings of Integrated Design and Process Technology, Society for Design and Process Science, June, 2008.

[2] Olewnik, A. T. and Lewis, K., *On Validating Engineering Design Decision Support Tools*, Concurrent Engineering, Vol. 13, pp. 111-121, 2005.

[3] Suh, N. P., *Axiomatic Design Theory for Systems*, Research in Engineering Design, Vol. 10, No. 4, pp. 189-209, 1998.

[4] Suh, N. P., Do, S., *Axiomatic Design of Software Systems*, CIRP Annals, Vol. 49, No. 1, pp. 95-100, 2000.

[5] Guenov, M.D., Barker, S.G., *Application of Axiomatic Design and Design Structure Matrix to the Decomposition of Engineering Systems*, Proceedings of Systems Engineering Wiley Periodicals, Inc., Vol. 8, No. 1, 2005

[6] Suh, N.P., *Axiomatic Design: Advances and Applications*, Oxford University Press, New York, 2001.

[7] Suh, N.P., *The Principles of Design*, Oxford University Press, New York, 1990

[8] Suh, N.P., Do, S., *Systematic OO Programming with Axiomatic Design*, Integrated Engineering, Vol. 32, Issue: 10, pp. 121-124, October, 1999.

[9] Huhns, M. N. and Singh, M. P., *Service-oriented computing: Key concepts and principles*, IEEE Internet Computing, 9(1):75-81, 2005.

[10] Gortmaker J., Janssen M., *Business Process Orchestration in e-Government: A Gap Analysis*, Proceedings of the 15th IRMA International Conference, New Orleans, LA, USA, 2004.

[11] Gortmaker, J., Janssen, M., Wagenaar, R. W., *The Advantages of Web Service Orchestration in Perspective*, Proceedings of the 6th International Conference on Electronic Commerce, ACM, Vol. 60, pp. 506-515, 2004.

[12] Akbıyık, E. K., *Service Oriented System Design through Process Decomposition*, Master of Science Thesis, Computer Engineering Department, Middle East Technical University, Turkey, August 2008.

[13] Wohed, P., Aalst, W.M.P.v.d., Dumas, M. and Hofstede, A.H.M.t.H, *Analysis of Web Services Composition Languages: The Case of BPEL4WS, Web Application Modeling and Development*, Conceptual Modeling - ER 2003, Springer-Verlag Heidelberg, 2003, 200 - 215.

[14] Toğay, C., *Systematic Component-Oriented Development with Axiomatic Design*, Doctoral Thesis, Computer Engineering Department, Middle East Technical University, Turkey, July 2008.

[15] Doğru, A.H., and Tanık, M.M., *A Process Model for Component-Oriented Software Engineering*, IEEE Software, Vol.20, No.2, pp. 34-41, 2003.

[16] Manzer, A., and Dogru, A.H., *Process Integration through Hierarchical Decomposition*, Proceedings of Enterprise Architecture and Integration, ISBN 978-1-59140-889-5 (eBook), pp.75-91, 2007.

[17] Papazoglou, M.P., and Georgakapoulos, G., *Service-Oriented Computing*, CACM, October 2003, 46(10).

[18] Kreger, H. et. al, *Management Using Web Services: A Proposed Architecture and Roadmap*, IBM, HP and Computer Associates, June 2005.

[19] Michael P. Papazoglou, M.P., Traverso,P., Dustdar, S., Leymann, F., *Service-Oriented Computing: A Research Roadmap*, International Journal of Cooperative Information Systems (IJCIS), Vol. 17, Issue: 2, pp. 223-255, 2008.

[20] Curbera, F., Khalaf, R., Nagy, W.A., Weerawarana,S., *Implementing BPEL4WS: the architecture of a BPEL4WS implementation*, Concurrency Computat.: Pract. Exper. 2006; 18:1219-1228

[21] Peltz, C., *Web Services and Orchestration and Choreography*, Proceedings of IEEE Computer Society, October 2003.

[22] Weerawarana, S., Francisco, C., *Business Process with BPEL4WS, Understanding BPEL4WS*, Part 1, research report, IBM developerWorks, August, 2002.

[23] Curbera, F., Andrews, T., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S., *Business Process Execution Language for Web Services Version 1.1*, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, 2003.

[24] Themistocleous, M., *Justifying the Decisions for EAI Implementations: a Validated Proposition of Influential Factors*, Journal of Enterprise Information Management, 17(2).

[25] Kreger, H., *Fulfilling the Web Services Promise*, Communications of the ACM, Vol. 46, Issue: 6, June 2003.

[26] Boh, W. F., Yellin, D.M., *Enablers and Benefits of Implementing Service-Oriented Architecture: An Empirical Investigation*, International Journal of Information Technology and Management, Vol. 9, Issue: 1, November, 2010.

APPENDIX A

ADSO DESIGN AND MODELLING TOOL

A.1 Adso

This procedure has begun with an extensive study on Adco tool provided in [14], and Sosecase tool provided in [12]. ADSO tool has been implemented to support ADSO methodology. ADSO tool has the capability to support a product line starting from the decomposition, and mapping of functional requirements with the design parameters, service composition, and graphical visualization of composition with SOSE language, and define activities in each process by using BPEL designer. The ADSO methodology proposes to support a product line that would include some capabilities of Sosecase, and Adco tool, as well as extending the coverage by automating the construction of SOSEML representation of process abstraction hierarchy from the design matrix. We have integrated the SOSE approach in Sosecase [12] with the Adco tool [14], while excluding the COSE approach in Adco [14]. On the other hand, Sosecase [12] was designed to form a decomposition tree from scratch. Since we have defined our design procedure with the help of ADT, we have used the modeling representation capability of SOSEML to visualize process abstractions in a hierarchy. The ADSO tool has the capability to create designs for software applications, and help to model the integration via a hierarchy view representation in SOSEML. There exist four views in ADSO tool:

- FR-DP design matrix
- Web service list
- SOSEML representation for integration support and decomposition visualization
- BPEL designer to model the process flow.

While constructing an FR-DP design matrix, the designer should select an existing web service interface from the existing web services. If a corresponding design parameter is not found, then the designer shall use the Web Service Design tab to contribute the new design parameter to the existing Web Service list via defining the same steps as the designer follows for application design.

A.1.1 Defining a new Web Service into the Application Domain

When the designer wants to add an non existing Web service to the system, she shall press the + button on the Web Service Design tab, and then the window appearing in Figure A.1 pops up on the screen.

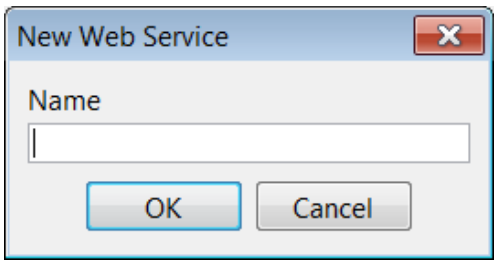

Figure A.1: The pop up window to insert the newly defined Web Service name

Then the new Web service name is listed at the end of the web service tree on the Web Service Design tab. You may refer to Figure A.2.

The designer shall double click the Axiomatic Design child below the newly added Web service name in the list depicted in Figure A.2, and the following abstract design matrix window is shown in the workspace panel of ADSO tool, Figure A.3.

When either row name, or column name is double clicked, the hierarchy of the FR and DP s is shown as in Figure A.4.

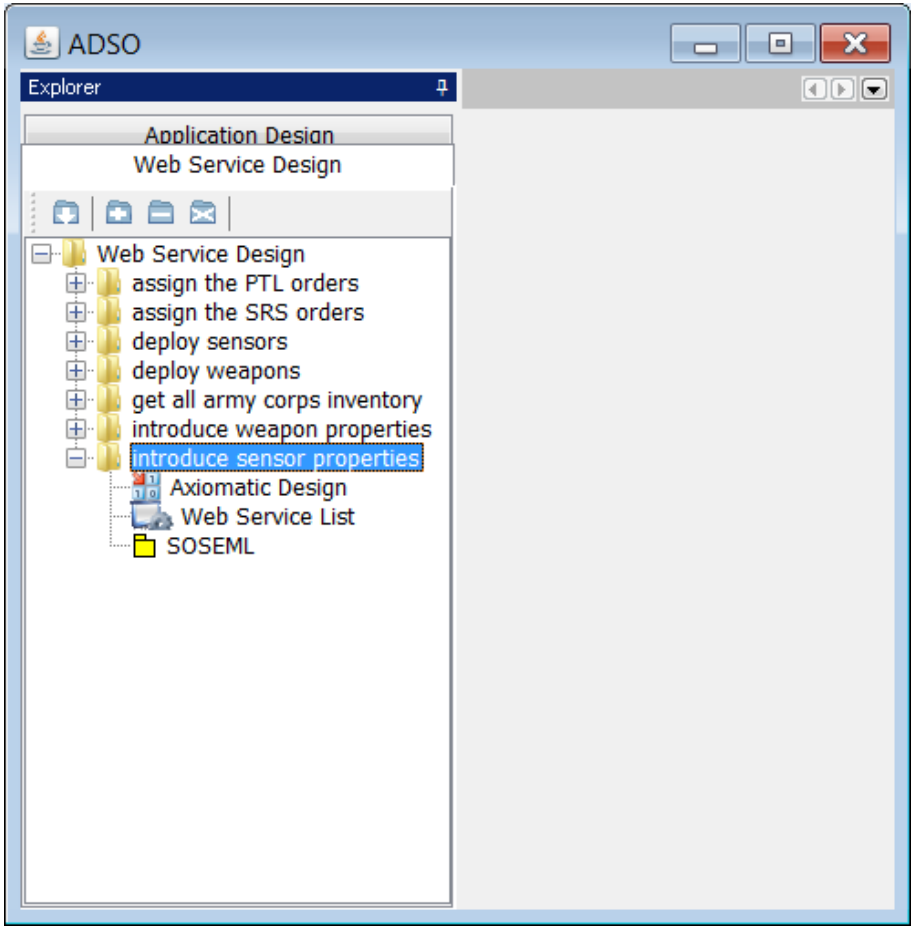

Figure A.2: A partial screen shot to visualize the existing web services depicted on Web service design tab

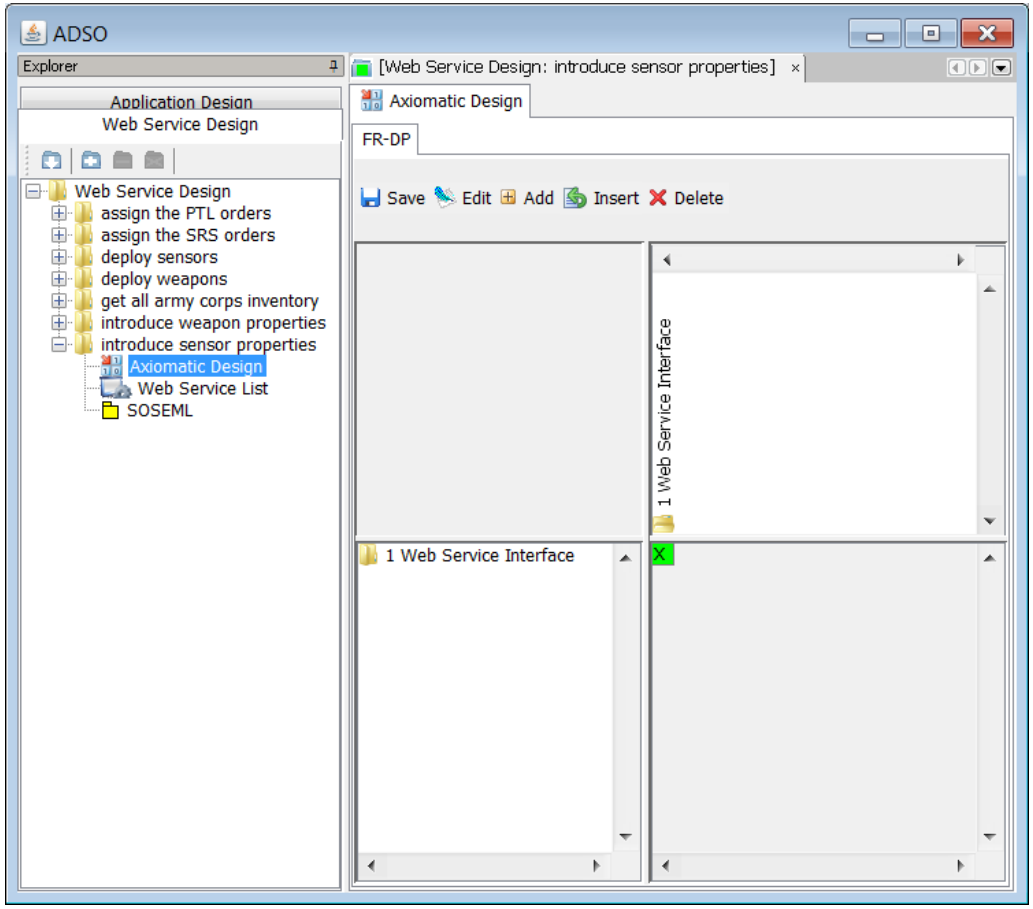

Figure A.3: The abstract design matrix for the newly added Web service

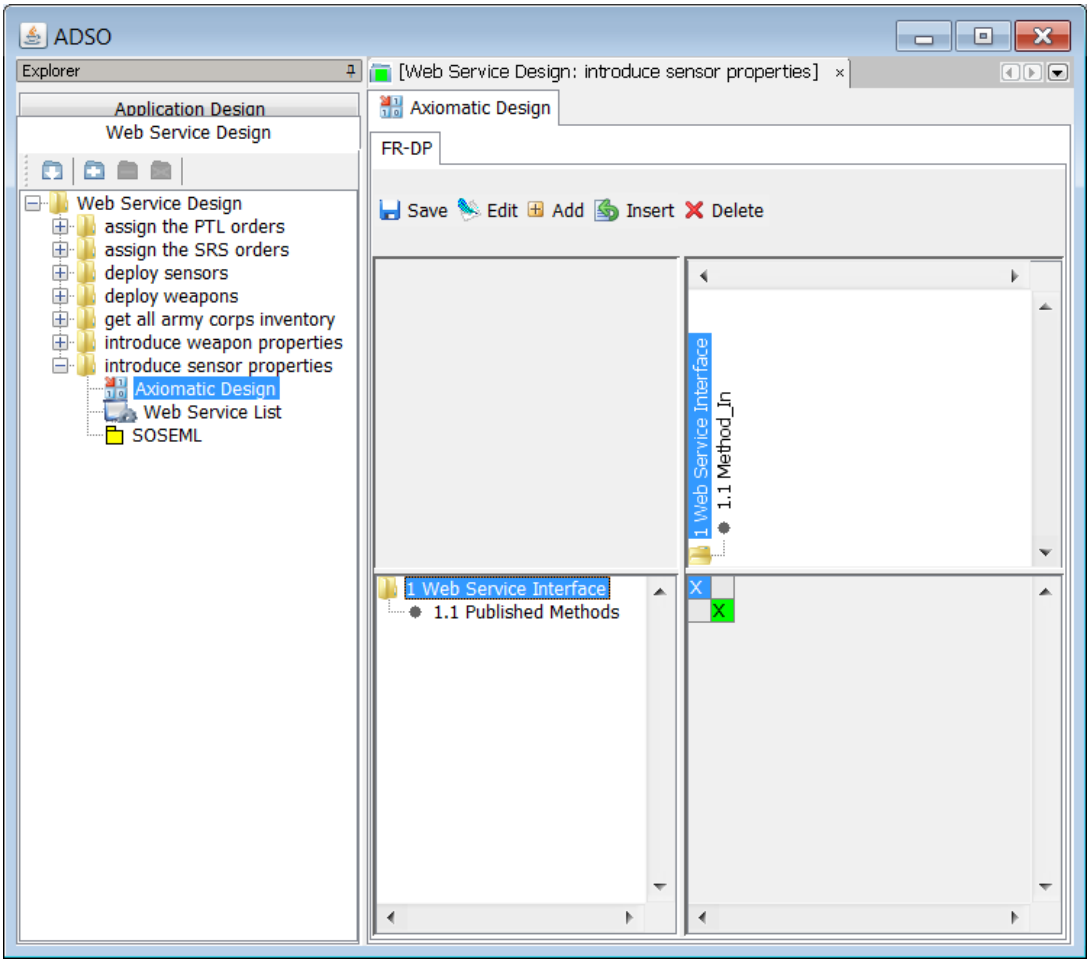

Figure A.4: Demonstration of FR and DP details in the design matrix

In order to add an FR, and its corresponding DP, once Published Methods name is chosen, Add icon should be pressed, and the user will be able to define the relational FR-DP couples by using the pop up window that appears on top of the ADSO tool.

When the user follows the order stated in Figure A.5, the new FR-DP couple will place on the design matrix.

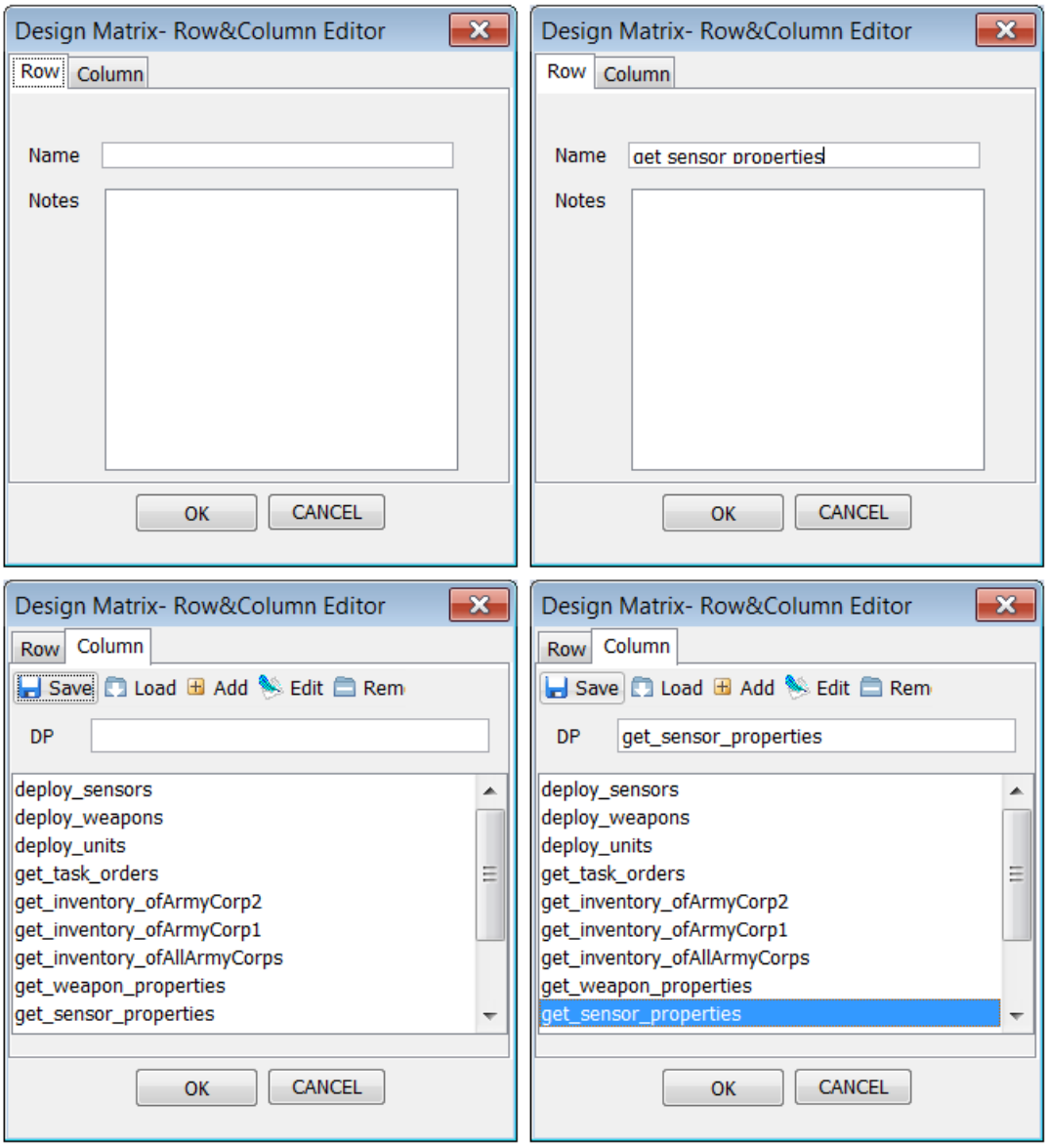

Figure A.5: Adding a new FR-DP couple to the design matrix

As shown in Figure A.5, the designer should type in the functional requirement, and some notes if wanted, then on the row tab, the designer may either find the already implemented Web service interfaces in the shown list, or add an interface name and click on Add icon, Load icon, and Save icon respectively. If the designer chooses from the web service interface list, all she has to do is click on Save icon, and click OK to exit the Design Matrix Row & Column Editor window.

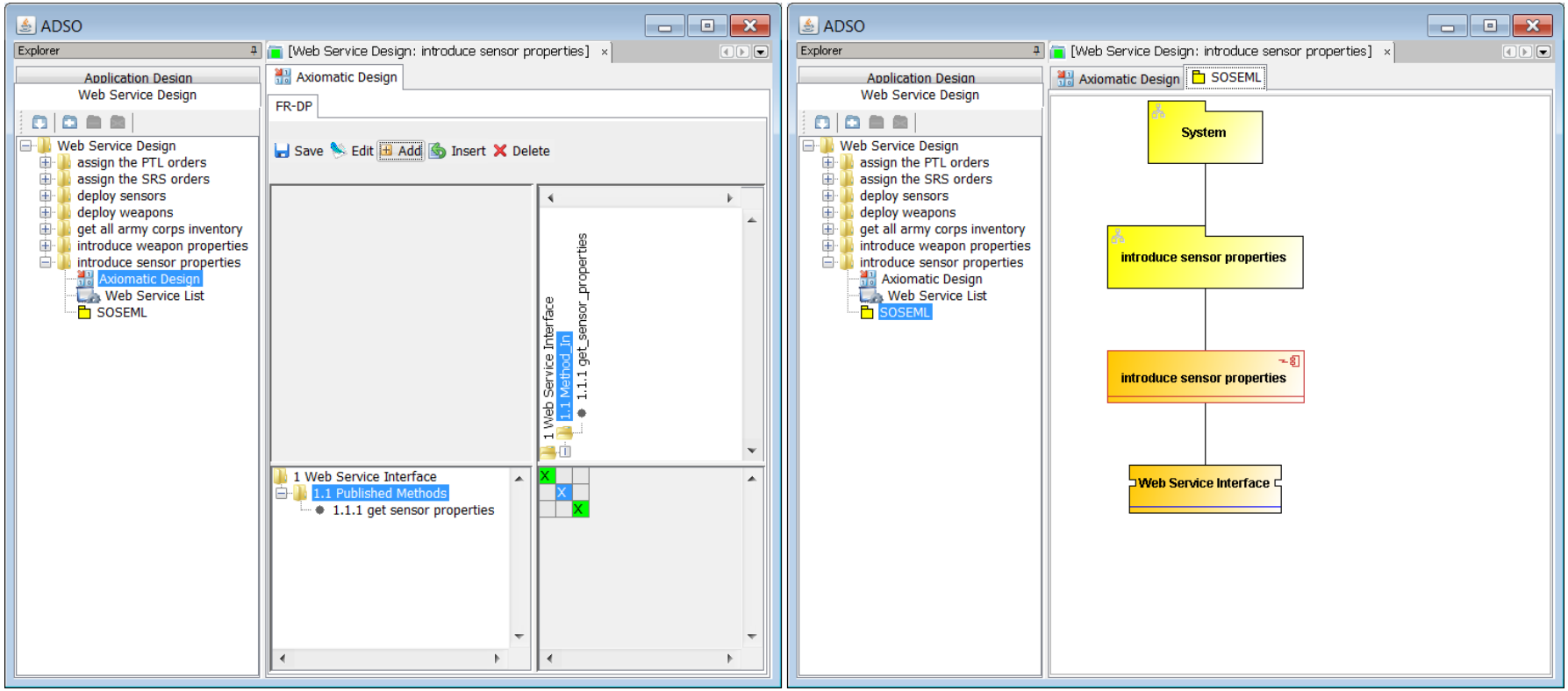

Figure A.6: The Web Service FR-DP design matrix and corresponding SOSEML representation

The corresponding SOSEML representation of defined Web service FR-DP design matrix is as in Figure A.6. As you see, the Web service is directly linked to the System process node, since Web service will be available for any process in the Application design. The corresponding Web service is depicted with its Web service name on the Web service node. The web service interface is also indicated with a link to its Web service, and the interface name is given on the web service interface. Each and every service shall be built by following the path explained in this section, since the ADSO methodology is for supporting reusability, yet expecting already defined and implemented Web services.

A.1.2 Constructing the Application FR-DP Design Matrix and SOSEML representation of the Application

After defining all the Web services that may be needed during the decomposition of the system, it is time to construct the FR-DP design matrix for the required system defining functional requirement. The application design tab is used during this procedure. First step to realize is typing in the FR-DP couples to the design matrix, and then the relations shall be evaluated for each FR, resulting in a design matrix of DPs covering more than one FR, and FRs being realized by more than one DP as in Figure A.7. If an FR is solved by many DPs, they have to be selected from the design matrix via clicking on the zero (0) fields, and choosing X from the drop downs.

This design matrix is evaluated in ADSO tool, and corresponding SOSEML representation is given by evaluating the Xs on the design matrix, meaning that the solution of the FR is supplied via X marked Web service interface's corresponding Web service. The SOSEML representation is given in Figure A.8, Figure A.9, and Figure A.10 in three parts, since the SOSEML representation is quite large to display on one screen.

A.1.3 Modeling Processes in BPEL Designer

Once the developer double-clicks any of the process figures, she may model the algorithmic process flow by taking the SOSEML representation tree into consideration, and handle the coordination of activities within the process by using the standard business process definition language in the BPEL designer. The BPEL models for the leaf level processes model the in-

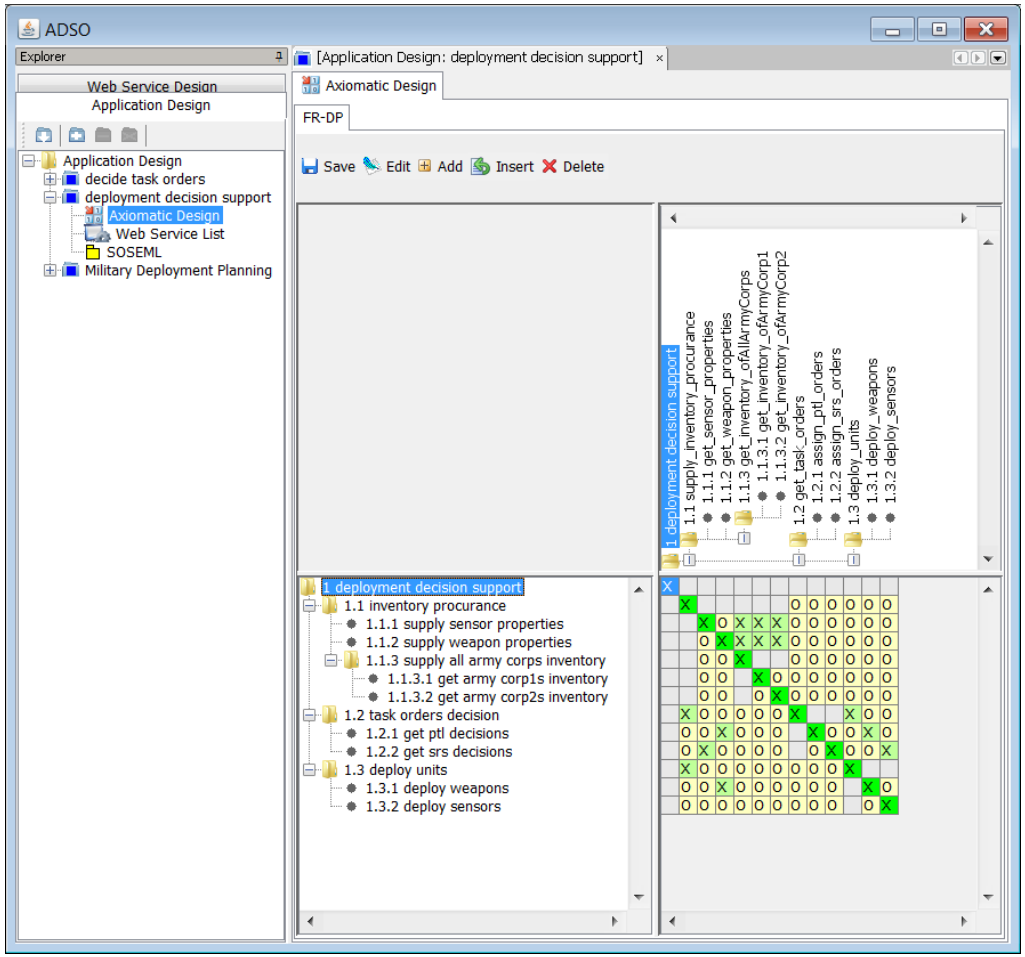

Figure A.7: Design matrix with all FRs coupled by related DPs for the Application

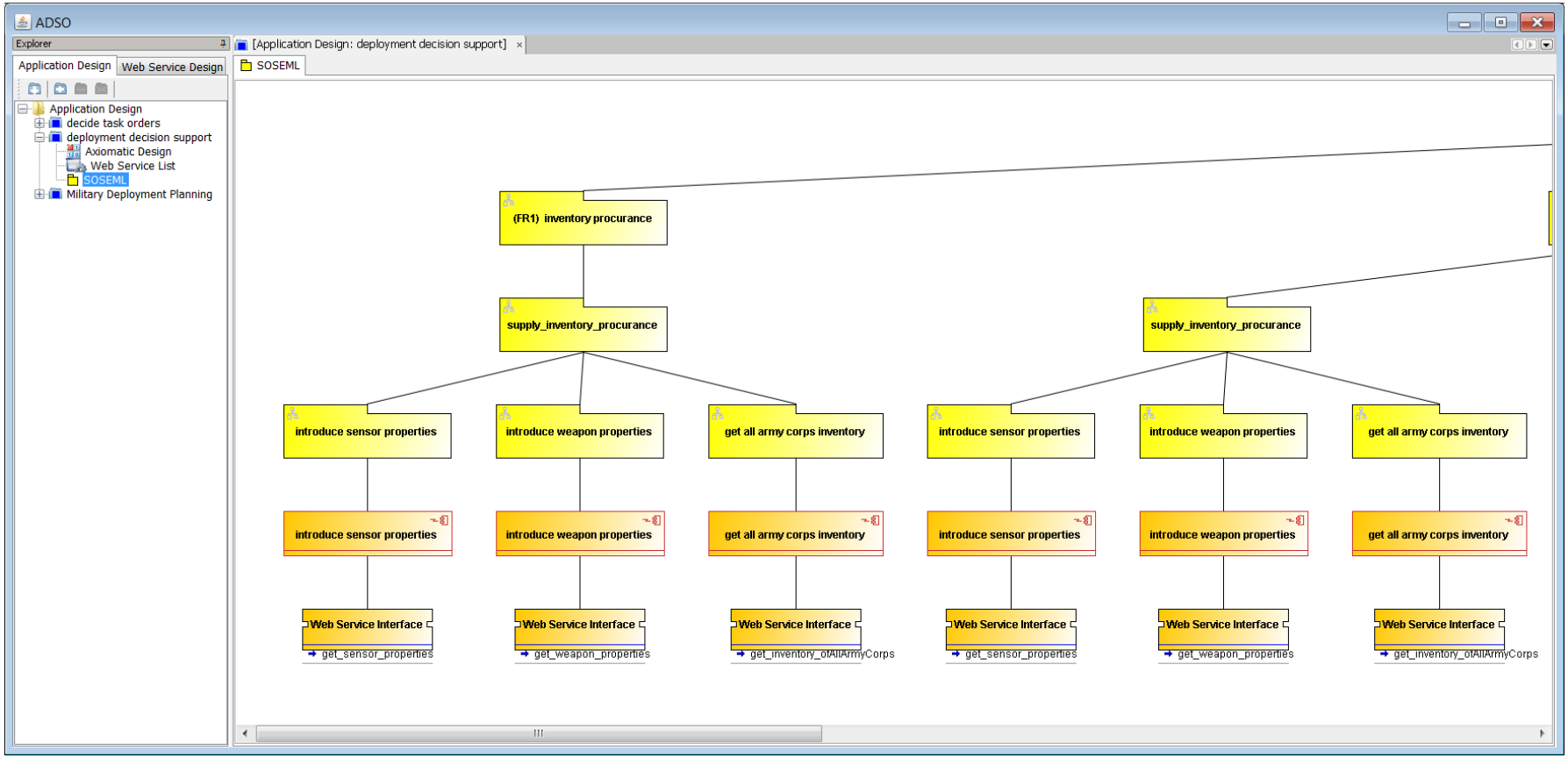

Figure A.8: Level 1 and 2 decomposition of the Deployment Decision Support Application sample

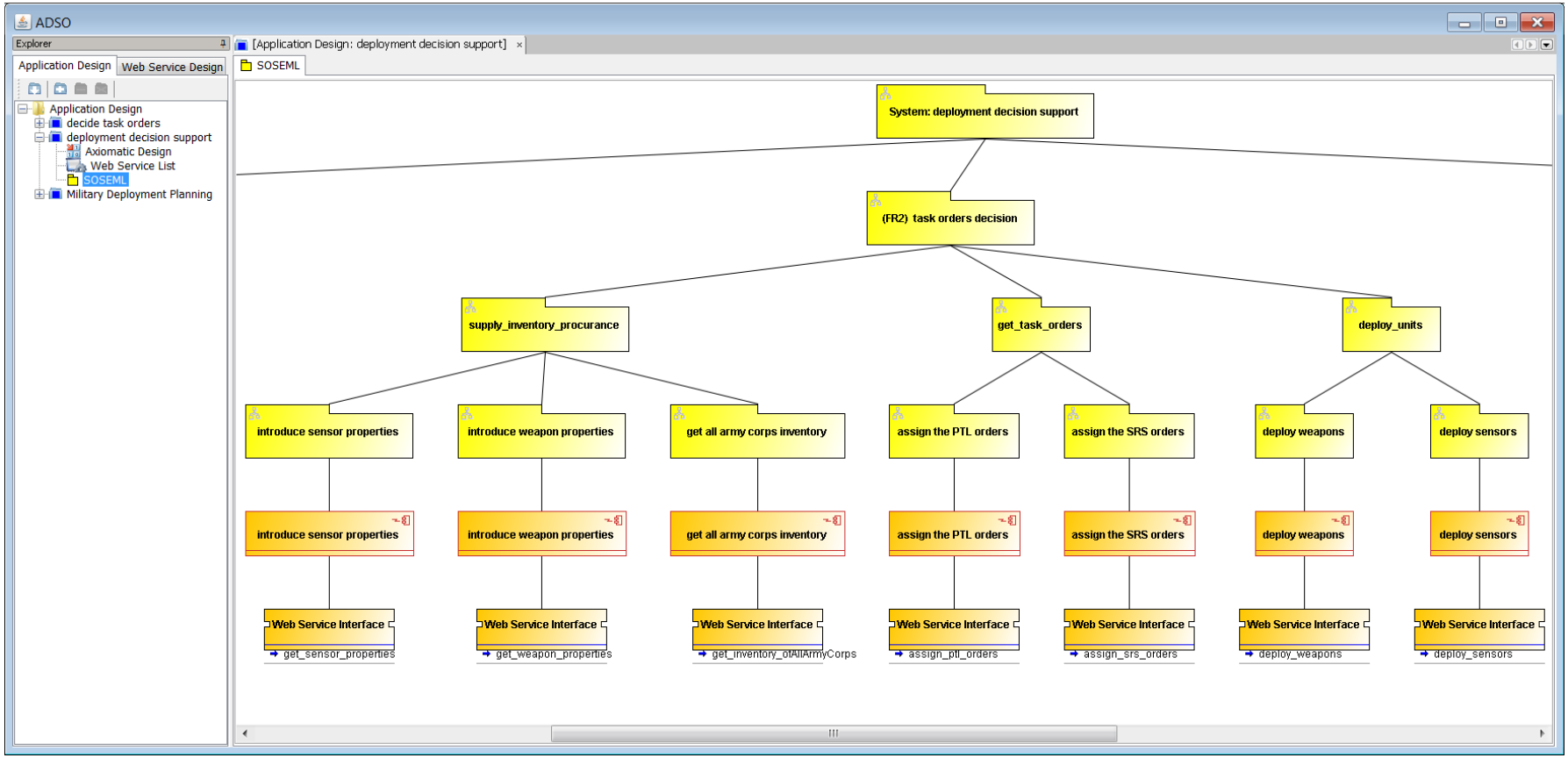

Figure A.9: Level 1 and 2 decomposition of the Deployment Decision Support Application sample (continued)

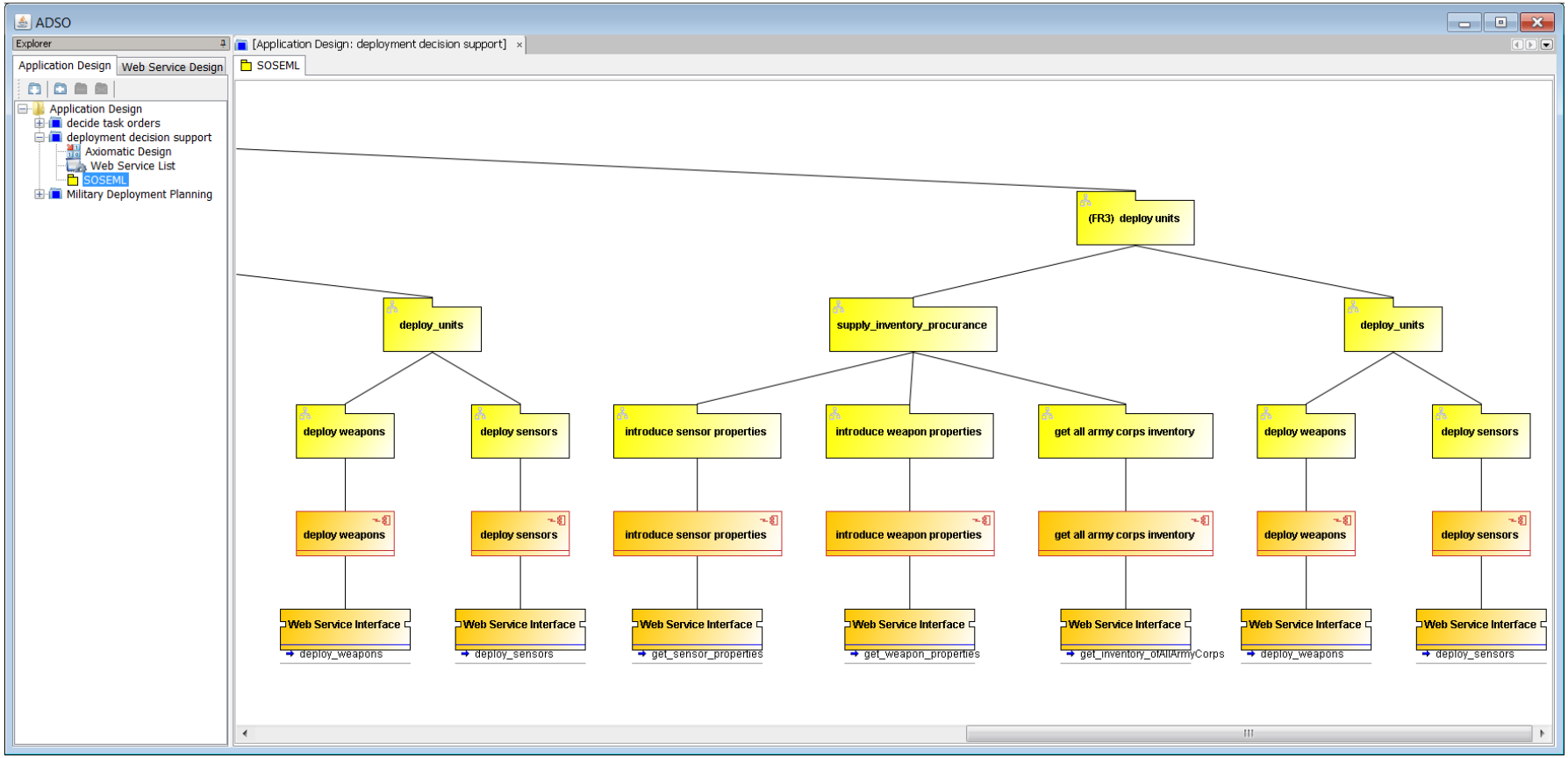

Figure A.10: Level 1 and 2 decomposition of the Deployment Decision Support Application sample (continued)

teractions among processes and existing Web services, while higher leveled processes assume lower level processes as the newly built Web services, and model the interactions among the processes, yet the whole system is defined in detail. In Figure A.11, the designer would like to model "introduce weapon properties" process, and the properties window is shown when the designer double clicks the process figure.

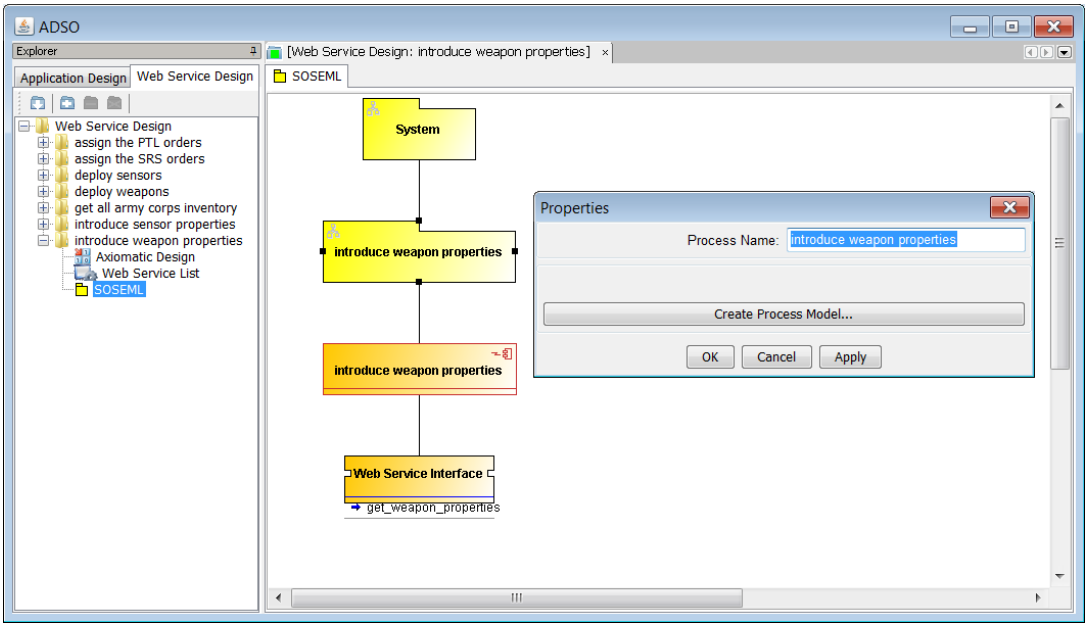

Figure A.11: The Properties window for a process with no existing bpel file

When the designer presses "Create Process Model..." button, the BPEL Designer, showing the relevant bpel file pops up. The original bpel file is in XML format, while this designer edits that XML file in graphical format as shown in Figure A.12.

When the process model is created, corresponding .bpel, and .wsdl files are obtained. The graphical representation of .bpel is as shown in Figure A.12, and can be edited to provide the actual corresponding process model, using the relevant introduce weapon properties service in this sample case. Meanwhile, the wsdl file can also be viewed in this designer, as depicted in Figure A.13.

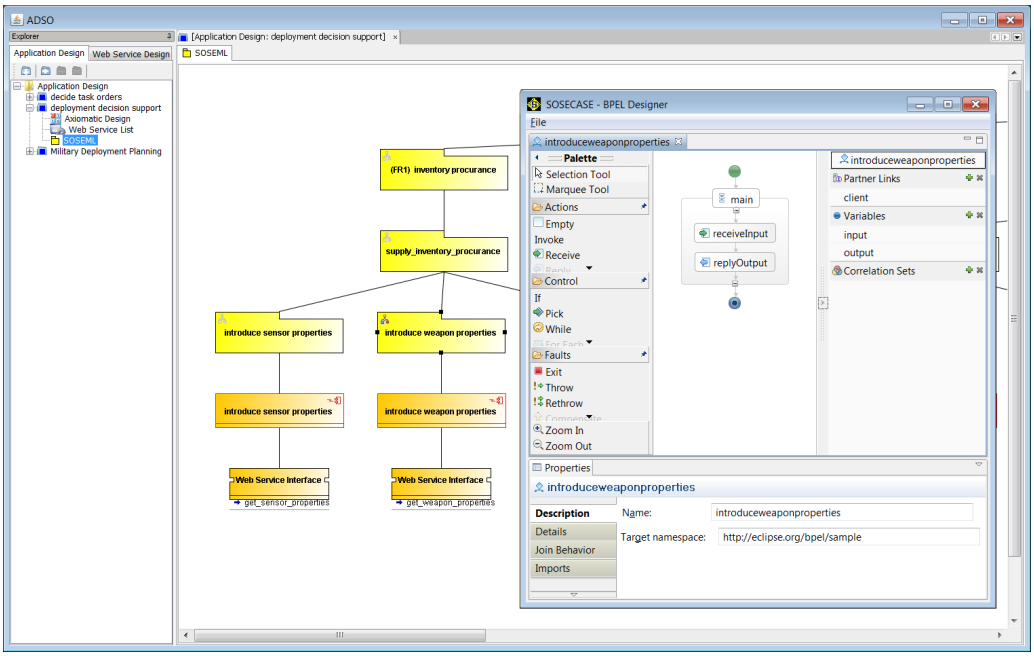

Figure A.12: The BPEL Designer showing “introduce weapon properties” process model, the process figure on SOSEML tree has its BPEL icon light on, indicating the process has an existing bpel file

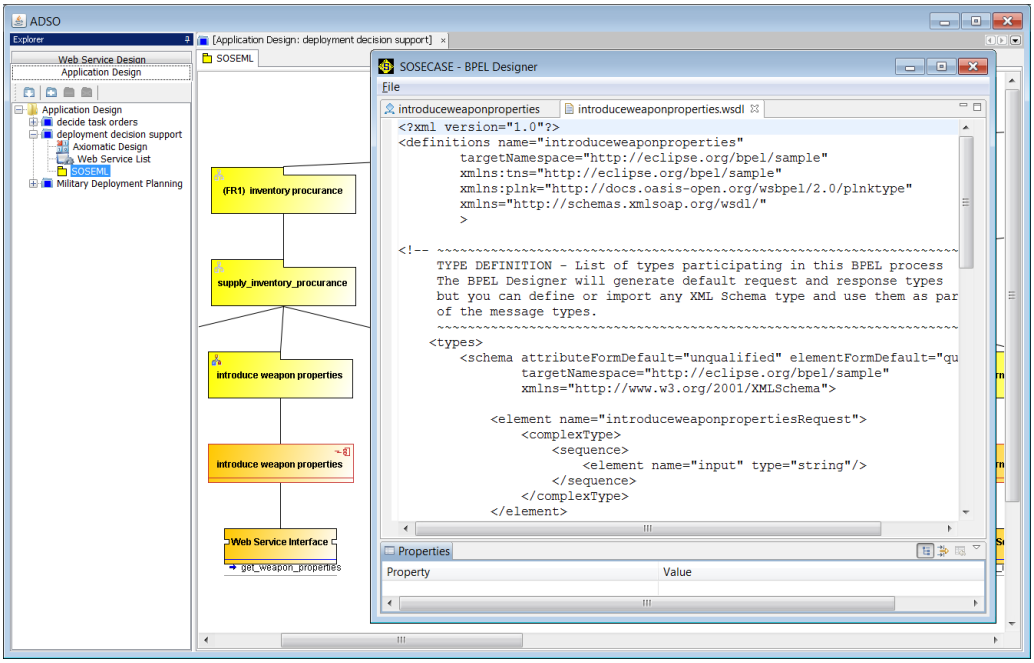

Figure A.13: The corresponding wsdl file for introduce weapon properties process model shown in BPEL designer.