International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 8, August - 2013

# Web Services Design From Business Requirements

Adel Qahmash
Department of Computer Science
Bridgeport University
CT, USA

Ayman Ibrahim
Department of Computer Science
La Trobe University
Melbourne, Australia

## Abstract

Web services have received a lot of attention from both researchers and the business IT sector. Most of web service research has been devoted to the issues of standards and supporting technologies. However, despite the considerable volume of research output, the actual application of web services for real business purposes has taken place much more slowly than expected: the number of business web-service-based applications in use is quite low. One of the main reasons for this slow take-up is the lack of research on designing web service applications from a business requirement perspective. Starting from the functional requirements (in terms of use cases), in this paper we propose a systematic method to specify the web service application that is designed to satisfy the functional requirements. The specification unambiguously shows how various parties, who are involved in the application, communicate with each other. Thus, the specifications clearly identify new web services to be built and how the new and existing web services can be composed to provide the required functional behavior. In addition, we also provide a systematic method to validate the specification through prototyping. The specification (which is platform-independent) and the prototype (which is in Java)

***Keywords; Web service, business rquirments, design-by-contract***



## 1. INTRODUCTION

Web service is a basic mechanism to describe, locate, and interact with online applications. Essentially, each application becomes an available web service component described by XML. According to the W3C [1], "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processed format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Web services have received much attention, especially in business-to-business (B2B) points of view, because of their high level of reuse and interoperability. Businesses and organizations connect to each other via the internet and the use of emerging web services to provide B2B interaction. These tools are built on top of existing web protocol and standard XML language. Web services composition facilitates the development of applications by reusing existing services [2].

Although there is a relatively large amount of research on web services, few effective commercial web services currently exist and engage in serious operations. Why?

On the one hand, a great deal of web service research has been devoted to the issues of standards and supporting technologies (including supporting composition from existing services). On the other hand, little attention has been devoted to the problem of designing web services on the basis of business-specified and business-perceived requirements.

More specifically, a critical issue can be stated as follows:Given the requirements of a business or a number of related businesses,

- How should we identify and build web services so that they can act as useful building blocks? and
- How can we determine the compositions needed for specific applications and be sure

that the behaviors of the compositions can be verified against business requirements?

Clearly, in order to gain an advantage from interoperable web services, we need to address this issue. This will be the aim of this paper. Our objective is to address this issue in a systematic and rigorous manner. Our method will refine the method introduced by [4] and will verify the method specification by implementing a prototype to insure that method design and specification is superior enough to capture business requirements.

For example, given a complicated "use case," such as the process of booking airline tickets and hotel accommodations with multiple communication exchanges between various parties (and some existing web services), how are we to

- Design web services and composed applications,
- Rigorously specify these services at the conceptual platform-independent level, and
- Develop ways (e.g. quick prototyping) to verify if our design satisfies the requirements of the "extended use case"?

The organization of the paper will be as follows: in section 2 a brief focus on related work will be examined. Section 3 a travel agent case study will be discussed. Section 4 presents a proposed method and design and analysis the method specification. Section 5 is concerned with the issue of the prototyping and implementation of the designed method. Finally, we will present our conclusions and possible directions for further work.

## 2. RELATED WORK

### A. *Designing web services with Tropos*

Tropos [2] was initially proposed as a method to design agent-oriented applications. Later, it was extended to become a service-oriented development method. A number of concepts, such as actor, goal, and social dependency, are introduced by Tropos methodology. This methodology has received a lot of attention because these concepts are described in such detail.

#### 1) *Tropos Phases*

1. **Early Requirements Analysis**. In this stage, the first step is to identify the stakeholders. The next step is to specify the goals, actors, and dependencies. Finally, the goals and tasks should be deconstructed to simplify the tasks. Additionally, during the early requirements stage, the system is disregarded.
2. **Late Requirements Analysis**. During the late requirements phase, the system, as well as its operational environment, relevant function, and qualities are introduced and discussed. Ultimately, the phase aims to associate each actor with its strategic goal.
3. **Architectural Design**. In this stage, the system's global architecture is specified in terms of actors. This specification defines subsystems that are interconnected through data and dependencies or control flow. Figure 6 shows how each extended actor diagram is produced to show how each subsystem is located in the whole system.
4. **Detailed Design**. During the detailed design stage, agent capabilities and interactions are defined. In addition, the detailed design phase aims to produce additional information for each architectural component of the system. Furthermore, using UML sequences, diagrams that model the interaction between agents can be recommended.
5. **Implementation**. In this stage, the actual implementation and code is carried out, depending on previous details set in the designphase.

Initially, we found this approach very helpful in our research. The Tropos methodology for designing web services starts from business requirements, which is of interest to our project. Moreover, the approach especially assists us in the early stage of web services design because of the significant attention it gives to business requirements.

However, the approach has disadvantages in that it contains many phases that do not precisely apply to our design. Tropos also does not describe how services should be integrated. Furthermore, Tropos approach does not include the support of

validation against requirements. Besides that, the most negative shortcoming is that Tropos fails to describe services in detail; it only introduces the phases to suggest their functionality.

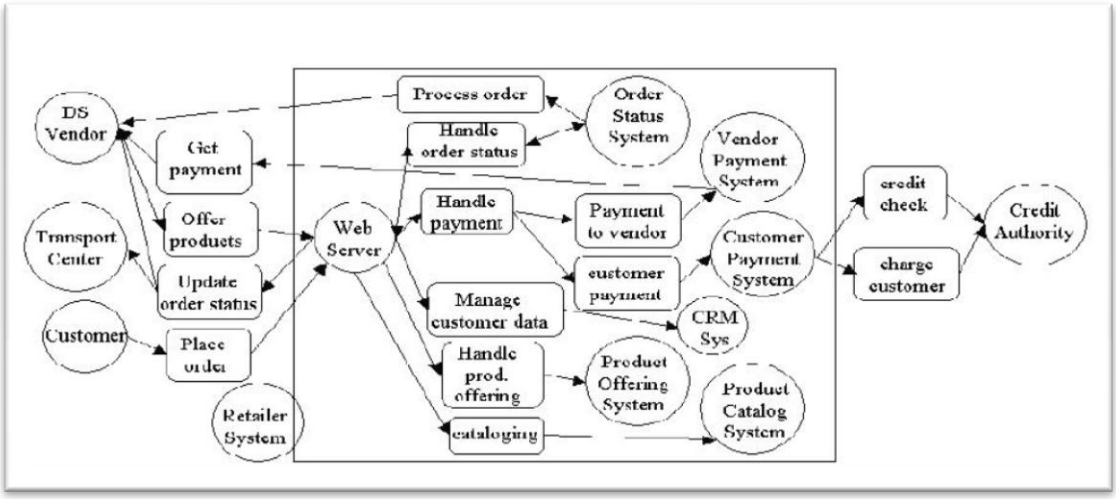


Figure 1. Extended actor diagram with respect to the Retailer System

## 3. TRAVEL AGENCY SYSTEM CASE STUDY

The Travel Agency System (TAS) is a modified version of the original case study conducted by [3], which is a process to sell tourism and travel services to the customer. In addition, the Travel Agency System is represented by a Web application conducting the process electronically.

For simplicity, this case study will consider just one type of transportation system- the airline system- and omit others types. Furthermore, various tourism services such as packages, accommodation and excursion that are afforded by the travel agent are not considered in this case study for the sake of simplicity.

As mentioned,TAS acts as an open distributed system that considers other services in order to satisfy the customer's request.

The ultimate goal of TAS that can support business aims is to sell a trip offer to the customer and consider TAS goal as use case.

The actors involved actors, along with ultimate goals of the Travel Agency System are as follows:

- Customers who can specify the trip information such as the destination along with other travel information.
- The personal travel assistant is the entity that can help the customer meet his requirements by booking flights and could be a software interface that interacts with customers. Hence, the customer request is managed by personal travel assistant until appropriate offer is found or the request is canceled by customer. In addition, to arrange the customer journey the personal travel assistant interacts with broker agents who may work with the travel agency.
- The Broker agent has 1-n relationship with the travel agency as well as with transportation companies. Moreover, the broker may access many transportation companies in order to obtain an appropriate flight itinerary.
- The transportation companies are entities that can provide actual transportation services. As mentioned before, airlines will be the only type of transportation systems considered in this case study.
- The financial companies are entities that can provide financial services like credit card companies as well as banks.

The Travel Agency System's operations are as follows:

- The customer should provide information about the desired trip trough the personal travel assistant. The trip information must consist of the departure and destination city as well as departure and return dates. In the case of a one way trip, the return date is not required.
- The trip information is received by the travel assistant system, which verifies

whether the information is well formatted. Then, brokers are selected by the travel assistant system to provide the customer's trip.

- Each broker agent may deal with many transportation companies requesting them to provide offers that match the requested trip. If the matched offer was provided, the broker will inquire with transportation companies to book the trip temporally, and send the matched offer (a long with corresponding overhead in the case of external broker) to the personal assistant.
- The personal travel assistant sorts matched offers, which were provided by all brokers. Then, the sorted list is send back to the customer to select the desired offer.
- The customer may select one offer or reject all offers and quit. In addition, the customer may refine trip information and start the process again.
- In case one offer is selected, then the credit card along with customer details should be provided to the personal travel assistant that will process the payment through one financial company. In addition, once the payment is confirmed, the travel assistant will notify the corresponding broker agent to confirm the booking via the corresponding transportation company. In case, the payment did not go through (insufficient funds, invalid credit card or expired credit) the customer will be able to either re-enter the credit card details or exit the application.
- For rejected offers, the personal travel assistant will notify the specific broker agent to cancel the booking through the appropriate transportation company.

## 4. BUILD THE SPECIFICATION OF ATOMIC USE CASE.

### A. *identfy the Atomic Use Case.*

The atomic use case can be identified if there is an operation that could be invoked from an external entity and possibly response messages could be received whether the response is asynchronous on not. Obviously, in a real Web service there is an operation that could be invoked from external services and those operations have a description files called WSDL that be accessible by the public. Thus, if we examine the behaviour of an atomic use case and Web service, we realize that there is a huge similarity between the two.

Figure 2 is a sequence diagram illustrating the use case scenario that just helps to identify the atomic use case. In this diagram, only the main flow is considered; the other sub flows are ignored.

There are three main atomic use cases, whichcan be derived from sequence diagram:

- The first operation is called "request offers" and starts when the customer enters the desired trip details into the system and ends when the system displays a set of matched offers to the customer.
- Another operation is called "reject all offers" and starts when customer rejects all offers, and the system sends a notification to each corresponding broker agent. The interoperable message in this operation is asynchronous.
- Finally, the place booking operation is started when the customer provides booking details to purchase the trip offer. The operation is finished when the customer receives a booking confirmation notification.

Driven by the above operations, the behaviors match those of atomic use cases. Thus, we will introduce the specifications of the above atomic use cases and will apply the proposed method in [4]. In addition, a *secondary input* and a *three party interaction building block* will be introduced to solve problems that arise.

#### 1) *Atomic Use Case Specification*

Initially, [4] proposed a method to specify atomic use case based on its input, output, pre-condition and poscondition specification, which will be applied to simply fulfill the requirements of usual application system.

However, in case of Web services there are number of factors that should be considered. Moreover, Web services obstacles may direct us to

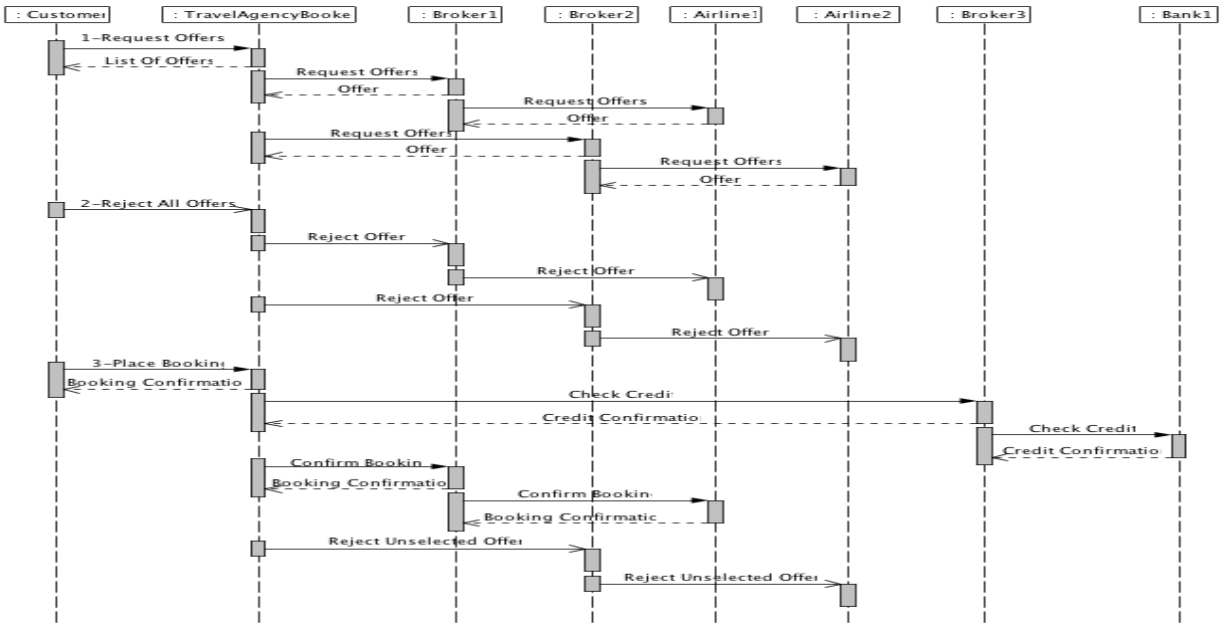


Figure 2. Order Trip Offer use case sequence diagram

introduce possible solutions such as *secondary input* and a *three party interaction building block.*

*2) Standard specification*

Assume that we have set of brokers called broker1 and broker2 and each of these brokers implements operations called getOffers using parameters inputs such as departureCity: String, destinationCity: String, departureDate: String, returnDate: String, isOneWay: boolean, and returns setOfOffers: set <Offer>. Consequently, atomic use case specification could be identified based on the proposed method in [4]. The specifications are as follows:

```
Atomic use case: Request offers
In:
departureCity?: String
destinationCity?: String
departureDate?: String
returnDate?: String
isOneWay?: boolean
out:
setOfOffers

pre:

// specified in details in the next
specification
post:
letsetOffers = new Set<Offer>
offer = broker1.getOffers(departureCity?,
destinationCity?, departureDate?,
returnDate?,isOneWay? )
setOffers.add(offer)
offer2 = broker2.getOffers(departureCity?,
destinationCity?, departureDate?,
returnDate?,isOneWay? )
setOffers.add(offer)
```

*3) Secondary Input Concept*

A number of problems that arise during above specification lead us to introduce the concept of secondary input to overcome those problems.

The first issue is how many brokers we will deal with. Thus, the above specification is not adequate enough to cope with this issue.

Another possible problem is that sometime we need to mange the number of brokers by adding or removing broker agents. In this case our specification needs to handle this matter independently with no need to with no need to modify the atomic use case specification when certain brokers are added or removed.

The third issue is that each broker provides an operation called getOffers that may accept different inputs or return different output. Hence, a sort of adaptation should be applied to avoid a broker having different specifications.

As designers, our concern is to specify atomic use case specification to a Web services developer who will be concernedwithWeb services implementation, dealing with brokers and sending them customer requests in order to obtain offers. Finally, the received information from external

brokers must be validated before using that information.

Consequently, introducing secondary input can assist the designer to overcome these problems effectively. In addition, secondary input can be distinguished from the ordinary input provided by user. Thus, the secondary input can be received by a system (i.e. set of offers for tip request) from an external entity (i.e. broker agents). To validate the secondary input, we specify secondary preconditions.

Furthermore, in terms of identifying the way we can interact with broker agents in order to receive secondary input, secondary input acquisition is specified to provide an effective guideline for Web services developers to invoke an internal operation of broker agent (i.e. getOffers). In addition, by using secondary input acquisition, we can specify certain information that provides inputs, outputs, collaborates, services that can be invoked along with a description of that secondary input. For example, as shown in the secondary input acquisition below, clearly we will deal with numbers of broker agents to invoke services called getOffers and send an inputs such as departureCity, destinationCity, departureDate, returnDate and isOneWay then received a set o offers as result of that invocation. Basically, the result of the above invocation can be known as secondary input in an atomic use case specification.

However, each broker may apply a different specification such as services name as well as inputs and outputs. In the implementation stage, we cope with this issue by specifying interfaces to be implemented by the brokers; thus, there is no different broker specification.

After introducing the secondary input, secondary input precondition and secondary input acquisition, specifying request off atomic use case will be more effective as follows:

```
Atomic use case: Request offers
In:
departureCity?: String
destinationCity?: String
departureDate?: String
returnDate?: String
isOneWay?: boolean
out:
setOfOffers
pre:
//departure and destination city is not the
same
departureCity?<>destinationCity?
//departure date is after system date
departureDate? >= sysDate
return date is after departure date for
//two ways trip
  ! isOneWay&&returnDate?  >= departureDate?
Secondary in:
providedOffers? = requestOfferFrombroker
(departureCity?, destinationCity?,
departureDate?, returnDate?, isOneWay? )
secondary pre:
//provided Offers from broker must be not
empty
providedOffers?.size> 0
//at lest one offer must match customer
request
//one way trip
exist offer in providedOffers? |
isOneWay&&
offer.departureCity! =departureCity? &&
offer. destinationCity!=destinationCity?
&&offer. departureDate! = departureDate?

//at lest one offer must match  customer
request
//two ways trip
exist offer in providedOffers? |
isOneWay&&
offer.departureCity! = departureCity? &&
offer.destinationCity! =destinationCity? &&
offer.departureDate! = departureDate?
offer.returnDate! = returnDate?
Post:
letsetoffOffers = new Set <Offers>
// return offer for  customer requested one
way
if(isOnWay)
then
for each offer in providedOffers |
offer. departureCity! = departureCity? &&
offer.destinationCity! = destinationCity? &&
offer.departureDate! = departureDate?
providedOffers.add (offer)
else
// return offer for  customer requested two
ways
offer.departureCity! = departureCity? &&
offer.destinationCity! = destinationCity? &&
offer.departureDate! = departureDate?
offer.returnDate! = returnDate?
providedOffers.add (offer)


Secondary Input
Acquisition:requestOfferFrombroker
in:
departureCity?: String
destinationCity?: String
departureDate?: String
returnDate?: String
isOneWay?: Boolean
out:
```

```
SetOfOffersFromBroker
Collaborate(s):setOfbroker
service: getOffers()
description:
Obtain offers from broker for a customer
requested trip.
```

To describe the process, the trip information inputs are received and checked by system. Then, an internal operation will be invoked to specify the secondary input. The result will be validated and finally add the matched offer to the set of offers. The customer will then receive that offer.

By above specification, reject all offers atomic use case can be specified in much detail in [11]

*4) Concept of Three-Party Interaction Blocks*

Introducing a three-party interaction building block technique (which may involves three different actors such as user, system and external party) will assist in building a third atomic use case specification. Place Booking atomic use case involves different complicated interactions with an external entity such as bank and a broker agent.

The Place Booking Atomic Use Case process will be as follows:

1. System requires further validation of the customer's credit card through the bank.
2. Based on the bank confirmation, if positive response was sent to the system, then the system will interact with the corresponding broker agent to book the trip.
3. Finally, the system will interact with the specified corresponding broker agent in order to reject offers that were not selected.

Obviously, a Place Booking Atomic Use Case can be split into three different internal operations, which are:

1. Confirm credit card, which interacts with external bank services.
2. Confirm booking, which interacts with broker agent.
3. Reject unselected offers, which interact with corresponding broker agents.

As we consider the above operations as part of the system, certain interactions between confirm credit card, confirm booking and reject unselected offers operations could be an important aspect of thePlace Booking Atomic Use Case. Thus, sometime we have to validate the result sent from operations to other or carry out certain tasks to ensure that the results are verified. Namely, offers should be filtered before sending them to broker agents or corresponding brokers. Each offer must be identified in order to reject unselected offers.

Obviously, the problem of specifying the interactions of sub-processes (confirm credit card, confirm booking, etc.) will be solved by applying the concept of three party interaction building blocks (interaction block for short) because using only secondary input will not overcome this dilemma. For clarification, the interaction block concept means that the Place Booking Atomic Use Case will be broken down into three interactions block that involve:

1. First block, confirming credit that starts when system receives the customer credit details, inputs those details and ends after receiving credit confirmation from bank.
2. Second block, confirms the booking that starts after receiving positive confirmation from the bank, then the system sends booking confirmation through the related broker agent.
3. Third block rejects unselected offers that start after confirming the booking, and then the system sends rejected offers to be processed by the related broker agent.

By examined above interaction blocks, certain general attributes can be used to recognize interaction blocks.

- Interaction blocks should receive inputs from users and inputs will be validated (preconditions).
- Interaction blocks will receive and validate the secondary inputs.
- Finally, further checking (postcondition) and sending respond to user will be performed by interaction blocks.

After introducing the interaction block, we can specify the atomic use case a long with its interaction blocks and the order of interaction that it uses. For example, assume there are two interactions blocks. We need to specify the order

of using them, supposing that the output of the earlier interaction block will be an input for the other interaction block.

Applying the original concept of the atomic use case companied with introducing the secondary input concept will assist in building the specification of interaction blocks.

**Interaction block:** Confirm Credit
**in :**
cardHolder? : String
creditCardNumber? : String
creditCardExDate? : String
paymentAmount? : Real
**out:**
creditConfirmation! : Boolean
**pre:**
// creditCardNumber? must be 16 digits
size(creditCardNumber?) == 16
// creditCardExDate? must be after today (system) date
creditCardExDate? >= System.Date
**secondary in:**
confirmCredit?: checkCrditCard ( cardHolder?,creditCardNumber?, creditCardExDate?,paymentAmount? )
**secondary pre:**
// confirmCredit? returns true
confirmCredit == true
**post:**
confirmCredit! = confirmCredit?

**Secondary input acquisition:** check credit card
**In:**
creditHolder: String
creditcardNumber: String
creditCaedExDate: String
**out:**confirmCredit: Boolean
**collaborate(s):** Banks
**Description:**
To verify the customer credit card through corresponding broker agent that interacts with related bank.

**Interaction block:** Confirm booking
**in:**
offer?: Offers
custName?: String
custPassportNo?: String
**out:**
confirmedBooking!: boolean

**pre:**
none
**secondary in:**
confirmedBooking? : confirmBookingToBroker(offer?, custName?, custPassportNo? );
**secondary pre:**
none
**post:**
confirmedBooking! = confirmedBooking?

**Secondary input acquisition:** confirm Booking To Broker
**in:**
offer?: Offers
custName?: String
custPassportNo?: String

**out:**
confirmedBooking!: boolean

**collaborate(s):** one corresponding broker
**services:**confirmBookingToBroker
**description:**
To call confirmBookingToBroker operation via one corresponding broker to confirm the booking

Finally, the last atomic use case, which is place booking, can be specified by its input and output and the order of the interaction block that occurred. The Place Booking Atomic Use Case specification is as follows:

**Atomic use case:** Place Booking
**in:**
confirmedOffer?: Offers
custName?: String
custPassportNo?: String
creditHolder?: String
creditCardNumbe?: String
pymrntAmount?: Real
**out:**
bookingConfirmation ! : Booking
Flow Description:

1. System calls checkCrditCard( cardHolder?, creditCardNumber?, creditCardExDate?, paymentAmount? ) and returns confirmCredit!
2. * System invokes confirmBookingToBroker(offer?, custName?, custPassportNo? ) And returns confirmedBooking!
3. * System will invoke rejectUnselectedOffers

```
(<list>AllOffers?: offer,
bookedOfferId?: String)
```

The indication of an asterisks (*) in the above specifications indicates that two processes can be carried out in parallel.

## 5. Implementation and Verification of the Design Method

In this implementation stage, we will implement the specification to ensure that our specification is validated, using Java SE 6. In this section we will presenta UML diagram for classes that will be implemented and then provide certain implementation code to explain how our method works.

Architecture for Travel Agency System includes airlines companies and banks that both are not our concerns just we will focus on the main system as well as broker agents. However, during implementation stage, external services such as airlines and banks should be considered just for implementation matters. The main class is TravelagencyBoker that interacts with set of brokers. In addition, we assigned particular brokers to deal with specific services for example broker1 can deals with airline1. Moreover, brokers who interact with airlines services must implement the airline broker interface as well as brokers who deal with banks must implement bank broker interface thus, each broker implements certain methods that exists in certain interface to standardize broker methods with our specifications.

Second important part of implementation stage will be simulation of SOAP messages to designed method parameters that will be sent to external services to unsure that sent information must be compatible with allowable SOAP data type. In general, SOAP allows certain data type to be included inside either request or respond envelope and supports primitive types (i.e. int, double), String and struct type and SOAP does not allow to pass whole object a long with its behaviors unless objects must be converted to one of acceptable data type or to serialize the object into XML.

## 6. Conclusion

In this paper, we presented a method to design, analyze and specify web service applications. Our starting point is the use case descriptions, which we employ as the main means of capturing the functional requirements. The specifications are formally specified. Besides widely-known elements such as inputs, outputs, preconditions, and postconditions, the specifications also make use of newly introduced elements such as secondary inputs, three-party interaction blocks; which are used to capture the communications that are specific to the web service nature of the applications.

We considered the case study of the Travel Agency System and applied our method to a comprehensive use case, which consists of several stages of interaction among the user, the Travel Agency System, and external service providers such as the airline and the banks.

By applying our proposed method, the resulting specifications are precise and complete with details to capture the full functionality given in the use case descriptions. A very important point about the specifications is that they describe, in logical platform-independent terms, the precise communication between various components of the systems, including the external web services. Consequently, the specifications allow us to identify the new web services that need to be created and how the web services collaborate with each other. In other words, the specifications are also a logical design for the composition of the web services.

We also provide a systematic method to create a prototype of the specifications. The method is simple and can be carried out with little cost. Nevertheless, the prototyping method can serve two useful purposes: (a) To validate the specifications; and (b) To demonstrate clearly how the specifications can be implemented in a chosen web service. The second purpose is achieved, in part, by ensuring that the messages to the objects representing web services are conformant to the data types and structures allowed by SOAP messages.

## 7. EVALUATION

As mentioned, Tropos approach is a method that has been most useful to our aims because this approach also starts to design web services from business requirements. However, as shown in Chapter 2, the Tropos approach has a number of disadvantages:

- Tropos produces specifications only at a very high level. A typical outcome of Tropos is the diagram shown in Figure 21. Such high-level specifications contain almost no details that are necessary to give the specifications precise meanings.
- Tropos suggests phases to support functionality of services without detailed descriptions of the web services and how the web services interaction with each other.
- Tropos produces models that are complicated (see Figure 1), which are hard to make sense of or to verify (How can we be confident that the design in Figure 21 would bring about the functionality that is required).

Compared with Tropos method, our proposed method has a number of desirable properties:

- Our approach is able to describe the specification much precise and provide detailed information about how we can implement the specifications.
- Our proposed method provides a clear guide for Web service developer to identify the web services, what they have to do individually, and how they should be composed to provide the intended behavior.
- Our proposed method produces specifications that are precise and can be validated against in order to ensure that they meet the functional requirements.

## 8. FUTURE WORKS

The proposed method is work in progress. In fact, it is very much in its initial development stage. There is much more investigation that needs to be done. Further investigations include the following:

Firstly, during stage of analysis and modeling, further analysis for system requirements and specifications should be performed by increasing the number of services provided by travel agency and external services to expand the method capacity.

Secondly, For Travel Agency System Case study (and others for that matter), additional requirements should be considered such as time factor and resource locking constraints (resource reservation). It is important to observe how these factors can affect the method and how the method should be expanded to deal with introduced factors.

In addition, further investigation can be made for the verification approach. We could, for example, provide a number of generic classes that can be used to prototype specifications more quickly and in a more standardized manner.



## REFERENCES

[1] W. S. A. working group "Web services architecture, w3c working group note 11 February 2004." access date 29/7/2008.

[2] D. Lau and J. Mylopoulos, "Design web services with tropos," in *the IEEE international conference on Web Services (ICWS'04),* July 2004.

[3] "A travel agency system, case study for workshop on model-driven web engineering (mdwe 2005), http://www.ice.uma.es/av/mdwe2005/thetasexample/." access date 15/8/2008.

[4] K. Nguyen and T. Dillon, "Atomic use case- a concept for precise modeling of object-oriented information system," in *The Ninth International Conference on Object-Oriented Information System*, (Geneva, Switzerland), 2003.

[5] A. Dennis, B. H. Wixom, and D. Tegarden, System Analysis and Design: An Object- Oriented Approach with UML. John Wiley and Sons, 2002.

[6] S. Bennett, J. Skelton, and K. Lunn, *Schaum's outline of UML*. McGraw-Hill, 2001.

[7] J. Hoffer, J. George, and J. Valacich, *Modern Systems Analysis and Design*. Prentice Hall,2005.

[8] T. N. Nguyen, *A Method for Analysis and Modeling of Web Service Applications and Their Compositions*, Honours Thesis. Department of Computer Science and Computer Engineering, La Trobe University, Bundoora, Victoria, 3086, Austraila, 2008. 14

[9] http://www.soapuser.com/basics3.html , SOAP user, Access Date 1-5-2009.

[10] J. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information system engineering: the tropos project," *information system journal*, 2002.

[11] A. Qahmash, *Web Service Design from Business Requirements*, Minor Thesis. Department of Computer Science and Computer Engineering, La Trobe University, Bundoora, Victoria, 3086, Austraila, 2009