Using Patterns in the Design of Inter-Organizational Systems - An Experience Report

John Novatnack[1] and Jana Koehler[2]

[1] Drexel University, Computer Science Department, Philadelphia, PA 19104, USA
jmn27@cs.drexel.edu
[2] IBM Zurich Research Laboratory, CH-8803 Rueschlikon, Switzerland
koe@zurich.ibm.com

Abstract. The modeling, design, and implementation of inter-organizational systems (IOS) is a challenging new problem. In contrast to previous systems, where components have clearly defined interfaces and serve a well-defined purpose, components of IOS exist in a distributed environment, where each component of the system may exist at a separate corporation and must conform to the interface of partner components. This creates a complicated problem for designers. In this paper, we investigate to which extent control flow, communication, and data manipulation patterns can help to ease the manual design process of IOS implemented in the Business Process Execution Language for Web services (BPEL4WS). By applying patterns of all three types in an iterative fashion we go from an abstract representation of the example process to a graphical solution with a one-to-one mapping to executable BPEL4WS code.

1 Introduction

The vision of Service-oriented architectures changes the way businesses are implemented and how they interact with each other. Business processes are no longer monolithic systems, but instead are assembled from existing service components. Companies interacting in a Business-to-Business (B2B) relationship as well as customers interacting with a company in a Business-to-Customer (B2C) relationship are no longer dealing with a single isolated corporation, they knowingly or unknowingly, interact with a composition of participating companies that work together to meet the customer's demands. The companies (and their business processes) participating in this composition form inter-organizational systems (IOS), consisting of independent and distributed services that come together to form a cohesive system.

The modeling, design, and implementation of these systems is a challenging problem. In contrast to previous IT implementations that had clearly defined interfaces and served a well-defined purpose, three trends of IOS can be observed: (1) Business requirements are much tighter linked to IT systems design, (2) IT systems are split into functional components (services) that have to serve very different purposes, and (3) the IT system boundaries break up - a system can no longer exist in isolation, but depends on various services which are offered by other partners.

These trends lead to a number of unresolved research problems [1]. One paramount problem is developing tools and techniques to design and implement these distributed systems. Established or upcoming standards such as the *Web Service Description Language* (WSDL) [2], which defines the public interface of a Web service, and the *Business Process Execution Language for Web services* (BPEL4WS) [3], which describes the operational details of Web service orchestration, provide a unique technological basis for the design of IOS. Knowledge of these languages alone does not equip a designer with the ability to design and implement IOS. Designers must make a conceptual leap from being fully authoritative over a system design, to designing a single synergic component in a collective system. Currently, little development support is available that aids the designer in this task. MDA techniques can be applied to UML sequence diagrams or activity diagrams to automatically generated executable BPEL4WS [4, 5]. Semantic Web approaches, e. g. [6], focus on the dynamic binding of Web services based on semantic annotations, which describe what a service does, under which circumstances it can be used, etc. The automatic composition of Web services is accomplished using intelligent planning techniques to achieve predefined goals. The approach requires to annotate individual BPEL4WS activities with pre- and postconditions in some logical language.

In this paper, we investigate to which extent *patterns* can aid in the *manual* design of IOS. More specifically, we investigate the role patterns play in the design of a Service-oriented Architecture based on WSDL and BPEL4WS and how patterns can be used to ease the development of the Web Service orchestration in BPEL4WS. While [4] focuses on *automatic*, one-to-one transformations from UML sequence diagrams to BPEL4WS code, but ignore details of data manipulation and communication amongst partners, we focus on *manual* transformations of UML sequence diagrams to BPEL4WS, where developers iteratively apply patterns to their design, all the while supplying the necessary data manipulation and communication details. As it has been observed in [6], “it is the BPEL4WS author’s responsibility to manually construct a process model that follows the operational semantics of tis service partners.” We investigate how iteratively applying patterns to a manual Web service design helps an author to assume this responsibility.

The paper is organized as follows. In Section 2, we briefly review the origins and properties of patterns and the state of the art in using patterns in implementing business processes. In Section 3, we introduce a specific development scenario and capture it with UML models. We start with formulating use cases that are subsequently refined in a UML sequence diagram showing the interactions between the partners of IOS. In Section 4 we begin transforming the UML sequence diagram into executable BPEL4WS code by injecting control flow, communication and data manipulation patterns. In Section 5 we draw conclusions and sketch some directions future work can take.

2 The Role of Patterns

The Merriam-Webster Online Dictionary defines *Patterns* as “a form or model proposed for imitation”. For the IT community, patterns have developed into “a software-engineering problem-solving discipline” which focuses on a culture to document and support sound design [7]. In general, there is wide agreement that defining what a pattern constitutes is hard. Instead, [7] declares a number of criteria shared by *good pat-*

terns. Following these criteria, a good pattern: (1) solves a problem, (2) is a proven concept, (3) proposes a nontrivial solution and (4) describes deep system structure and relationships.

The most popular patterns in software development originate from the Object-Oriented Community [8]. They were later extended to patterns that address general issues of software architecture [9] as well as concurrent and networked objects [10]. At the same time, patterns of control-flow were extracted from workflow systems [11].

2.1 Patterns for IOS

When designing IOS based on a choreography of Web services, three areas where patterns may be applicable can be identified:

1. *Control flow* between the various Web services,
2. *Communication* between a number of partner Web services, and
3. *Data manipulation* that is implemented in the choreography.

The workflow patterns defined by Aalst et al. [11] have been extracted from the control flow constructs provided by workflow systems. The patterns have been mainly used to classify and compare existing workflow products based on the control flow constructs that these products provide, and have recently been extended to compare proposed Web service standards [12]. There is, however, no work done up to this point that examines the impact and the applicability of the workflow patterns in the design and implementation of IOS. A set of communication patterns have been defined that may have relevance to the Web service domain [13]. The communication patterns are separated into two broad categories, patterns dealing with (1) synchronous and (2) asynchronous communication. A set of patterns for enterprise integration is given in [14]. In our study, we focus on the more simpler set of communication patterns of [13] as we are especially interested in differentiating between the synchronous and asynchronous communication as it occurs in BPEL4WS, and aiding in applying each. Recently, work has begun in the workflow domain to develop a set of data patterns [15].

3 A Case Study of Manual Incremental Pattern-Based Design

In our case study we bring three different trends together: Service-Oriented Architectures based on WSDL and BPEL4WS, Model-driven Architecture [16], and Patterns. Models play an increasing role in the design of software. In the following, we create models, expressed as UML use case and sequence diagrams, to capture the design of a simplified Web-based travel agency process. We use models to capture the major requirements and then inject patterns into our models in order to move towards executable BPEL4WS code. Thus in this approach, patterns are considered as guides that facilitate the refinement of models by an IT expert. We believe that some automatic tooling support may be available in the future to partially automate the pattern injection process (similar to applying OO Design Patterns in Rational XDE for example), but this issue is beyond the scope of this paper.

We study the applicability of patterns in the context of a specific example, namely a travel agency. The travel agency receives requests from customers over the phone. The requests are recorded in trip documents, from which several flight and hotel requests are derived by the agency. The flight and hotel requests are handled by independent Web services that exist outside and receive requests from the travel agency. The services report their success or failure in booking the requests back to the agency. If all requests were successfully handled, the agency puts together the trip documents for the customer and confirms the travel itinerary. If one of the services fails to make a requested reservation, the agency informs the other service to put all reservation requests on hold and contacts the customer to revise the trip itinerary. Our design goal is to arrive at a BPEL4WS specification that implements the interaction of the travel agency with its partners. In this paper we are not concerned with the internal details of the flight and hotel services, although the techniques described here can also be applied to their design.

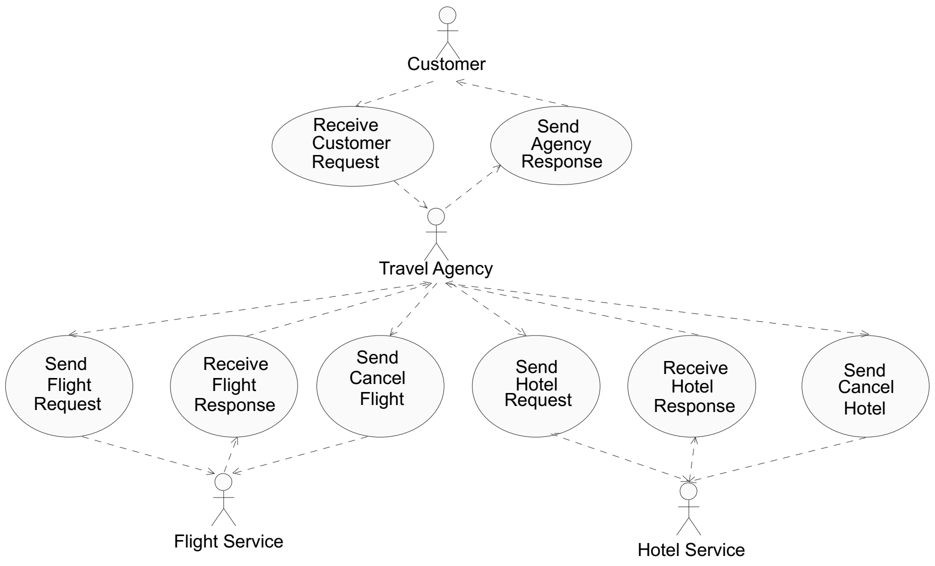

Fig. 1. Behavioral view of the travel agency process as a use case diagram.

As was stated in [17], Web services may be modeled from two views: *collaborative* and *behavioral*. In the collaborative view, IOS are modeled from the perspective of an external observer. The observer views the sequence of messages exchanged between Web service partners in the system. In the behavioral view, individual Web services are independently modeled from an internal perspective that treats partners like a black box. In this paper we model the travel agency from an internal, or behavioral, view. However, if the behavioral models of the customer, flight and hotel services were available, it would be possible to construct a collaborative view of the composite system. Figure 1 captures the main use cases of how the four partners, customer, travel agency, flight service, and hotel service, interact, modeled from the behavioral view of the travel agency.

Use case diagrams provide a high level view of the partner interaction. It is a natural next step to use a UML sequence diagram to further refine the required interaction between the partners, cf. Figure 2. Since our implementation will be based on BPEL4WS and WSDL, the Web service messages which are exchanged in a specific order help in further understanding the design of the travel agency process. Figure 1 uses so called *co-regions* (depicted with bold brackets) from UML2 to specify that some messages can be sent in any order. Conditions in braces are added to the diagram to capture that the CancelFlight and CancelHotel messages will only be sent under certain conditions.

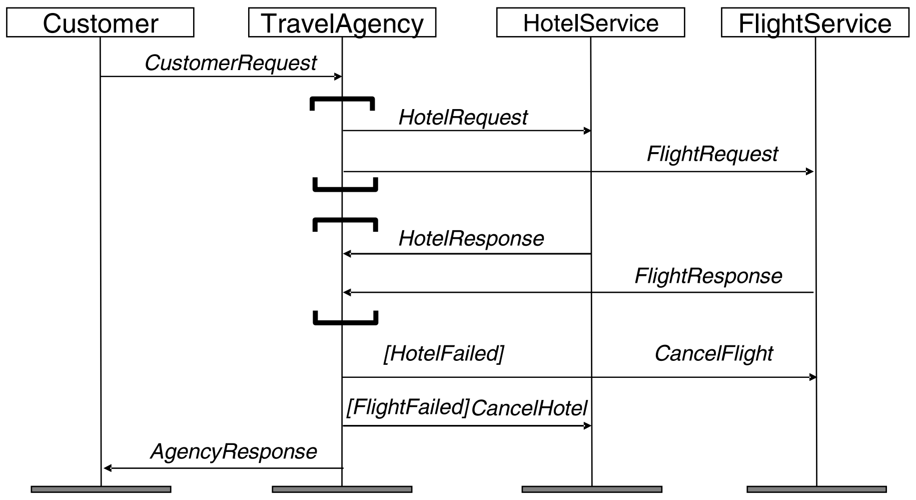

Fig. 2. Sequence diagram for the travel agency process

4 Applying Patterns to the Design

Further refinements are made to the sequence diagram by injecting control flow, communication and data manipulation patterns. The injection of these patterns allows the design to approach executable BPEL4WS code.

4.1 Injecting Control Flow Patterns

The set of workflow patterns that we consider are divided into six categories [11]:

1. Basic control
2. Advanced branching
3. Structural patterns

4. Patterns involving multiple instances
5. State based
6. Cancellation

Each category contains patterns that supply certain control flow constructs. In general this includes two types of patterns: patterns that initiate a sequential or concurrent split in the control flow and patterns that merge a number of sequential or concurrent flows into a single chain of execution. Although captured in individual patterns in the above collection, it is often the case that the merge patterns are provided implicitly by BPEL4WS, depending on how the split is implemented.

Figure 3 shows places where the control flow patterns can be applied to the design of the travel agency. In designing and implementing the travel agency process, patterns from basically any of the categories can be applied. With extensive knowledge of the patterns and the system to be implemented, a designer can make a choice of the most applicable patterns. Animations from the workflow patterns web site [18] provide an intuitive basis for designers to choose which patterns to apply. However, there does not exist classification system as comprehensive as provided by the OO community, where patterns are listed along with *intent*, *motivation*, *applicability*, *consequences* and *known uses* [8].

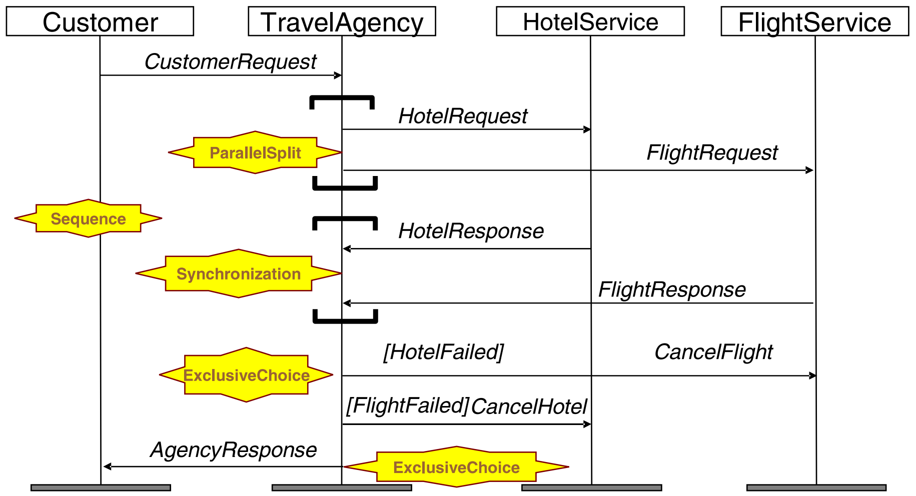

Fig. 3. Placing control flow patterns in the sequence diagram.

We decide to apply several patterns from the set of basic control patterns, namely the *sequence*, *parallel split*, *synchronization*, and *exclusive choice*. The *sequence* pattern executes activities in order, where one activity begins after the previous activity has completed. This pattern is appropriate for the interaction between travel agency and customer. The *parallel split* pattern allows for the execution of a number of activities in parallel. It is applied to design the concurrent interaction of the travel agency with its partner services. The *synchronization* pattern is used to merge the parallel split. Following the synchronization pattern, the travel agency needs to make a choice depending on whether the partners report a success or failure. The *exclusive choice* pattern is applied to provide a conditional branching construct.

All four of these patterns have a direct mapping to BPEL4WS code, see also [12], where the sequence pattern is implemented using a `<sequence>` element, parallel split and the synchronization patterns are jointly implemented using a `<flow>` element and exclusive choice is implemented using a `<switch>` element. Alternative mappings for all patterns can be provided to the designer by using `<flow>` elements with control links that supply transition and join conditions.

4.2 Injecting Communication Patterns

Web service communication can be distinguished as being either *synchronous* or *asynchronous*. In synchronous communication, the sender halts for a response after sending a message, whereas in asynchronous communication, a sender does not implicitly block for a response after sending a message to a partner. BPEL4WS naturally supports both forms through the constructs: `<invoke>`, `<receive>`, `<reply>` and `<pick>`. We consider a set of communication patterns which are organized into synchronous and asynchronous patterns [13]. The synchronous patterns include: (1) Request/Reply, (2) One-Way and (3) Synchronous Polling. The asynchronous patterns include: (1) Message Passing, (2) Publish/Subscribe and (3) Broadcast. Of the synchronous communication patterns we apply *Request/Reply* and of the asynchronous patterns we apply *Message Passing*. In the synchronous Request/Reply pattern a sender exchanges a message with a partner and blocks for a response. In the asynchronous Message Passing pattern a sender exchanges a message with a receiver without expecting a response.

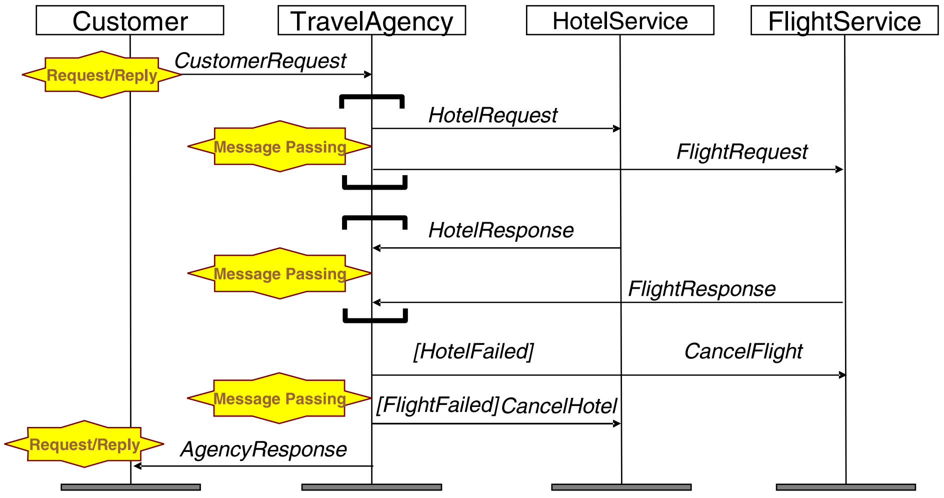

Fig. 4. Placing communication patterns in the sequence diagram.

Figure 4 shows the UML sequence diagram of the travel agency with the injected communication patterns. Determining which communication links will be synchronous and which will be asynchronous is up to the designer of the system. A customer which interacts with the travel agency expects that the travel agency will respond to the reservation request. The customer and travel agency engage in a synchronous exchange of message, where the customer halts and waits for the travel agency's response to the reservation request. The travel agency, on the other hand, may have the capability to interact with an arbitrary number of flight and hotel service partners. The responses given by some of the partners may have long delays, or may never come. Asynchronous communication links between the travel agency and flight and hotel service partners allows for controlled communication, where one lost message will not block control flow within the process. Secondly, when the travel agency sends a CancelFlight or CancelHotel message, it does not expect a response, therefore a synchronous message exchange would cause the travel agency to block infinitely. Asynchronous message passing is *required* here, as there is no return message.

The Request/Reply and Message Passing patterns are both implemented using the BPEL4WS communication constructs: `<invoke>`, `<receive>` and `<reply>`, or `<receive>` and `<pick>`, see also [12]. The synchronous Request/Reply pattern may be implemented by the sender with an `<invoke>` element that specifies an input and output variable, and with a receiver that uses the `<receive>` and `<reply>` elements. The asynchronous Message Passing pattern is implemented by a sender that uses an `<invoke>` statement with only an input variable and a receiver that blocks with a `<receive>` element. If the initial sender requires a response to an asynchronous message, it is necessary for them to block with a `<receive>` element.

4.3 Injecting Data Manipulation Patterns

A good starting point for data manipulation patterns are the patterns originally designed by the OO community [8]. These patterns are very well explored and provide a basis for constructing the data manipulation patterns necessary for Web services. As a starting point, we define the *mediator*, inspired by the OO pattern of the same name. In the OO domain the mediator pattern is "an object that encapsulates how a set of objects interact" [8]. With respect to IOS, the mediator pattern encapsulates how a set of *partner services* interact. Web services exchange complex XML variables that may be instances of WSDL message types or defined by XML schemas. Collaborating Web services are independently designed and cannot be expected to have conforming interfaces. It is up to the Web services participating in the collaboration to manipulate data such that it conforms to the interface definition of each partner. The mediator pattern encapsulates the data manipulation that must be done in order for the data to be sent to partner services in a valid format.

Figure 5 shows an abstract view of the mediator pattern. The pattern is initiated with a complex message containing multiple data components going to independent partners. The mediator pattern splits the message into new messages, each conforming to the interface definition of a partner. Correlation data is added when it is necessary to distinguish between multiple instances of a single partner type. The mediator pattern

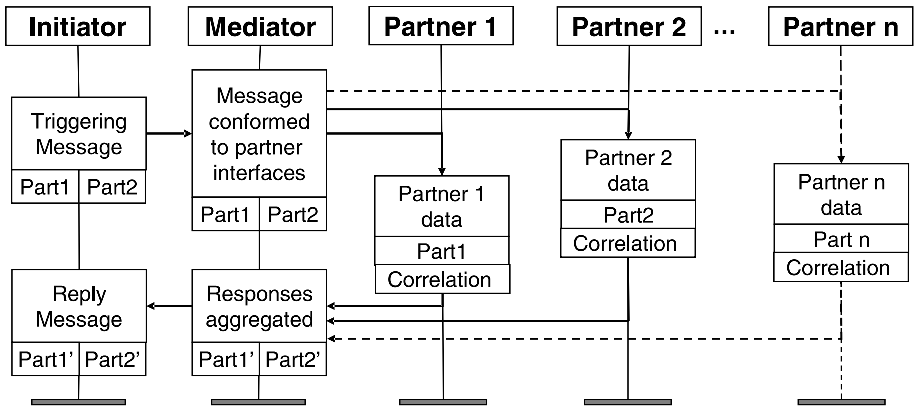

Fig. 5. Abstract view of mediator pattern

receives responses from the partner services and aggregates the modified data into a format suitable for the initiator of the pattern.

A relevant portion of the mediator pattern's sample code in BPEL4WS is shown in Figure 6. The BPEL4WS code breaks the pattern into three sections: first a receive activity obtains the data from the triggering partner. Secondly, the data is split into two separate variables with an assign activity. Synchronous invoke statements send the messages and receive responses from the service partners. Finally, the output of the partner services is combined into a single message that is returned to the initially triggering partner. The sample code shows essential details of how messages are bound to variables and how variables are copied and assigned to each other. In addition, correlation sets may need to be added to the communication constructs to distinguish between multiple instances of partner services. Similar to the description of the OO patterns, sample code of sufficient detail needs to be provided.

4.4 Resulting BPEL4WS Flow

Figure 7 shows the graphical representation of the resulting BPEL4WS code of the travel agency obtained after successful application of the patterns, based on the UML profile for BPEL4WS [19]. The graphical representation can be seen as *executable*, as it has a one-to-one mapping to BPEL4WS code. The travel agency begins a sequential execution of activities after a `<receive>` element obtains the initial synchronous request from the customer. The travel agency asynchronously invokes the flight and hotel partner services in parallel with a `<flow>` element (depicted as a thick horizontal bar) and implements the mediator pattern to handle the data manipulation required for the interaction, with an `<assign>` element. The synchronization pattern is applied at the end of the parallel execution, and an exclusive choice, implemented with a `<switch>`, is done to determine the results of the reservation request. The travel agency returns a synchronous reply to the customer with a `<reply>` element.

```
<!-- Initial message reception and data breakup -->
<receive variable="triggerMessage" partnerLink="initiator" ... />
<assign>
  <copy>
    <from variable="triggerMessage" part="part1" />
    <to variable="partner1Message" part="part1" />
  </copy>
  <copy>
    <from variable="triggerMessage" part="part2" />
    <to variable="partner2Message" part="part2" />
  </copy>
</assign>

<!-- Communicating with independent partners -->
<invoke inputVariable="partner1Message"
        outputVariable="partner1Output"
        partnerLink="partner1" ... />
<invoke inputVariable="partner2Message"
        outputVariable="partner2Output"
        partnerLink="partner2" ... />

<!-- Aggregating the data and returning a response -->
<assign>
  <copy>
    <from variable="partner1Output" part="part1" />
    <to variable="triggerMessage" part="part1" />
  </copy>
  <copy>
    <from variable="partner2Output" part="part2" />
    <to variable="triggerMessage" part="part2" />
  </copy>
</assign>
<invoke inputVariable="triggerMessage" partnerLink="initiator" ... />
```

Fig. 6. BPEL4WS sample code of mediator pattern

Bringing the three patterns together in a consistent manner has allowed us to transform the initial UML use case diagram into an executable model. Throughout the application of the patterns it is necessary for the designer to supply the details of the variable access and control flow decisions. Variables in a BPEL4WS process can be instances of a WSDL message, or defined by XML schema. Data manipulation patterns, such as mediator, must be supplied with the structure of these variables in order to access the correct components. In addition, a designer must explicitly define the Boolean propositions that will drive control flow decisions. Both of these tasks can be simplified through the use of design tools that provide an intuitive graphical interface.

Our approach of incremental design combining patterns that address different aspects of an IOS is similar to the approach described in [20]. As Janson et al., we start with the analysis of a business scenario and then refine it by investigating the collaboration and interaction between the partners in more detail. In contrast to Janson et al. the patterns we consider, provide control flow, communication, and data manipulation solutions, which are independent of a specific implementation, while their patterns focus on available ready-to-use software components, e. g. web client systems, adapters, workflow engines that are plugged together.

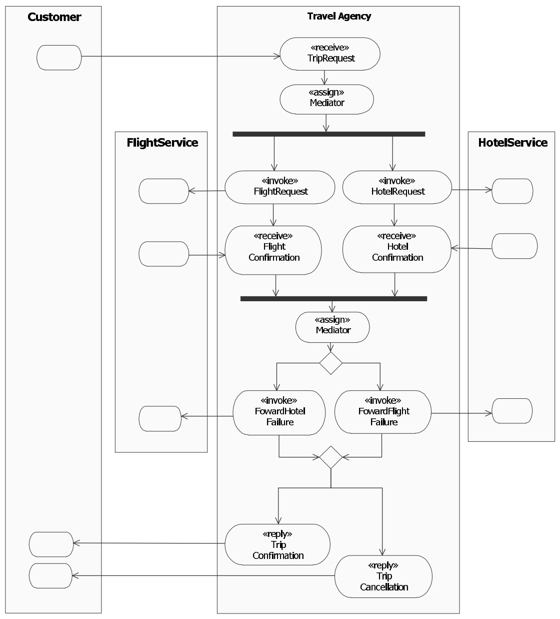
Customer
Travel Agency
«receive» TripRequest
«assign» Mediator
FlightService
HotelService
«invoke» FlightRequest
«invoke» HotelRequest
«receive» Flight Confirmation
«receive» Hotel Confirmation
«assign» Mediator
«invoke» FowardHotel Failure
«invoke» FowardFlight Failure
«reply» Trip Confirmation
«reply» Trip Cancellation

Fig. 7. The abstract BPEL4WS resulting from the pattern application.

5 Conclusion and Outlook

In this paper we have combined techniques of Model Driven Architecture and reusable design techniques to transform an abstract view of a Web service to a graphical representation with a direct mapping to executable code. This was accomplished by applying *patterns* from three important aspects of Web service design: control flow, communication and data manipulation. Prior work has enabled a starting point for creating a set of control flow and communication patterns appropriate for Web service design. Data manipulation patterns still need to be extensively explored. The application of patterns to the design of a Web service is only beneficial when the patterns are applied in a way that is consistent in all three areas. For example, the application of a synchronous communication pattern will affect how the mediator pattern communicates with partner services.

Our work has shown the elements exist to apply Web service patterns to the design of IOS. The next step in the development of patterns for IOS must include a standardized method of classifying patterns, such that designers are aided in applying them. Similarly to the experience of the OO community, these classifications will not be possible until there is a larger amount of experience in the design of IOS. The identification of problems and the knowledge of creating solutions to these problems will allow more sophisticated Web service composition patterns to be created that address all three aspects of the system design in a consistent manner.

References

1. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: A look behind the curtain. In: Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS). (2003) 1–14
2. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: The web services description language WSDL. http://www-4.ibm.com/ software/ solutions/ webservices/ resources.html (2001)
3. Andrews, T., et al.: Business process execution language for web services. www-106.ibm.com/developerworks/webservices/library/ws-bpel/ (2002)
4. Bauer, B., Mueller, J.P.: MDA applied: From sequence diagrams to web service choreography. In Koch, N., Fraternali, P., Wirsing, M., eds.: Web Engineering: 4th International Conference (ICWE), Lecture Notes in Computer Science 3140 (2004) 132–136
5. Hauser, R., Koehler, J.: Compiling process graphs into executable code. In: Third International Conference on Generative Programming and Component Engineering. LNCS, Springer (2004) forthcoming.
6. Mandell, D.J., McIlraith, S.A.: Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In: International Semantic Web Conference 2003. (2003) 227–241
7. Coplien, J.O.: A pattern definition. Hillside.net (2004)
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of reusable object-oriented software. Addison–Wesley (1986)
9. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture - A System of Patterns. John Wiley and Sons (1996)
10. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. John Wiley and Sons (2000)
11. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B.., Barros, A.: Workflow patterns. In: Distributed and Parallel Databases. (2003) 5–51
12. Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A..: Analysis of web services composition languages: The case of BPEL4WS. In: 22nd International Conference on Conceptual Modeling (ER 2003). (2003) 200–215
13. Ruh, W., Maginnis, F., Brown, W.: Enterprise Application Integration: A Wiley Tech Brief. John Wiley and Sons, Inc (2001)
14. Hohpe, G., Woolf, B., Brown, K., DCruz, C.F., Fowler, M., Neville, S., Rettig, M.J., Simon, J.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley (2004)
15. Russell, N., ter Hofstede, A.H., Edmond, D., van der Aalst, W.M.: Workflow data patterns. White paper, Centre for Information Technology Innovation (2004) http://www.citi.qut.edu.au/babel/DP/DP.jsp.
16. The-Object-Management-Group: OMG model driven architecture. http://www.omg.org/mda (2003)
17. Frank, J.H., Gardner, T.A., Johnston, S.K., White, S.A., Iyengar, S.: Business processes definition metamodel concepts and overview. White paper, IBM Software Group (2004) http://www.omg.org/cgi-bin/apps/doc?bei/04-05-03.pdf.
18. : Workflow Patterns. (2004) http://www.workflowpatterns.com/.
19. Amsden, J., Gardner, T., Griffin, C., Iyengar, S., Knapman, J.: UML profile for automated business processes with a mapping to BPEL 1.0. IBM Alphaworks http://dwdemos.alphaworks.ibm.com/ wstk/ common/ wstkdoc/ services/ demos/ uml2bpel/ docs/ UMLProfileForBusinessProcesses1.0.pdf (2003)
20. Janson, D.H., et al.: Patterns: Designing Process Integration Solutions. ibm.com/redbooks (2004)