Chapter 11

High Assurance BPEL Process Models

Mark Robinson, Hui Shen, Jianwei Niu

Department of Computer Science, University of Texas at San Antonio

One UTSA circle, San Antonio, Texas, 78249

Abstract. An increasing number of software applications and business processes are relying upon the use of web services to achieve their requirements. This is due in part to the use of standardized composition languages like the Business Process Execution Language (BPEL). BPEL allows the process designer to compose a procedural workflow from an arbitrary number of available web services and supplemental "programming-like" activities (e.g., assigning values to variables). Such composition languages naturally bring concerns of reliability, consistency, and durability, let alone safety and security. Thus, there is a need for formal specification and analysis of BPEL compositions for high assurance satisfaction. We propose the use of Unified Modeling Language (UML) sequence diagrams as a means for analysis of BPEL process consistency and demonstrate our technique with two examples.

1. Introduction

Today, web services play an important role in service-oriented computing, as the web is an inarguably ubiquitous medium. Web services provide domain-specific functionality to client and server applications alike, with the interface to those services exposed over the web. There are many web services currently available and incorporating a web service into an application is simple, although the integration may impose a reasonable learning curve. Creating a web service is also simple, with many different development platforms already equipped to produce web services.

Web services offer numerous advantages to both the web service consumer and the web service provider. The use of web services affects the software engineering process in many advantageous ways. Some of these advantages are:

J. Dong et al. (eds.), *High Assurance Services Computing*,
DOI 10.1007/978-0-387-87658-0_11,

- Fast and cheap to deploy – while these advantages are not unique to web services, it is worth stating that reductions in time-to-market and cost-to-market are real benefits from utilizing web service technology. Web service providers specialize in their domains and realize economies of scale by making their services affordable to web service consumers. Additionally, the use of the Internet as a communication medium reduces cost.
- Reusability – one can easily reuse the same web services in new applications, no matter which development language is being used (e.g., PHP, JSP, .NET). The only requirement is that the development language provides support for accessing web services.
- Accessibility – many different platforms and devices may utilize the functionality of web services, including mobile devices. Client-side applications, ad-hoc queries, and web sites/applications may all access the same web service (see Figure 1). The web service provider may also freely control how the service is accessed and how a service consumer is charged for its use. This allows for a variety of governance and payment options for the web service, more easily allowing it to fit the budgetary constraints of the service consumer.
- Centralized method of discovery – web service providers can register their web services with online registries that provide descriptions of the web services provide. A description may also include pricing for the use of the service and a link to the interface specifications required to communicate with the web service (e.g., WSDL document).
- Maintainability – the provider of the web service is responsible for maintaining, securing, and updating the web service and its data. The web service consumer bears none of the labor for these tasks, although it is normal for the web service provider to pass on some of the cost for maintenance to the web service consumers.
- Value-added content – integration of services that provide proprietary functionality and/or data may increase the value of the web service consumer's product, particularly if the consumer is acting as a service broker.
- Loose coupling – the consumer of the web service does not care how the provider of the service implements the service, provided its functionality is known, consistent, and reliable. Future changes to the service or switching web service providers will not necessitate changes to the consumer's application as long as those changes do not affect the service interface.

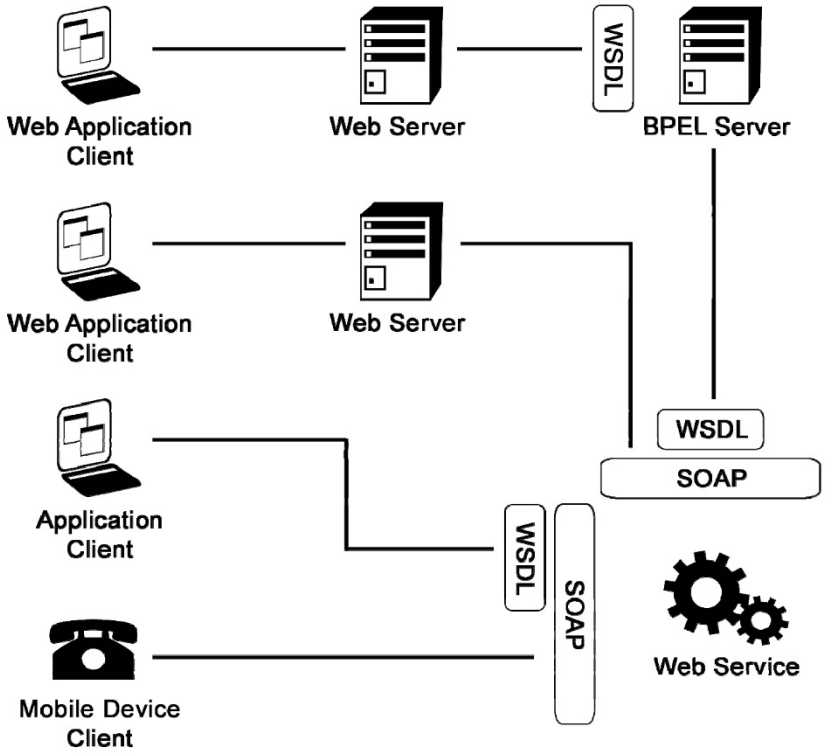

Fig. 1. Various devices and platforms accessing the same web service.

Integrating web services into a software project can be a frightening notion, as the developer might begin to consider some of the negative possibilities of such integration: loss of service, price gouging, data security and privacy, etc. But, these concerns have always existed in component-based software engineering. Trust and long, solid track records can help ease fears of integration disaster. But if those are in short supply, contracts and service level agreements can be tailored to suit both parties' concerns and project requirements. Web service providers and consumers should also consider Business Interruption and Errors and Omissions insurance to protect each party in the areas where a contract does not.

A web service is essentially a program that is located on a web server and has its interface exposed to the outside world using Internet protocols (of which the normal "web" comprises just a few). The web service interface is constructed in a standardized fashion using a technology like the Simple Object Access Protocol (SOAP). A web service performs a very specific function, or set of functions. The functions typically suit a single domain, e.g., a product catalog. Expected function input and output has to be provided to potential consumers in order for the potential consumers to know how to address the interface. This can be accomplished us-

ing web service directories, developer documentation, or Web Service Definition Language (WSDL) documents. A WSDL document is an Extensible Markup Language (XML) description of the interface to the web service. Many web service client technologies can use the WSDL document directly to utilize the web service. WSDL documents simplify the interface aspect of using web services. Using technologies like SOAP or WSDL, the task of integrating a web service into an application (e.g., a web site) is straightforward.

1.1 BPEL

There is a further abstraction of web service technology known as the Business Process Execution Language (BPEL). BPEL is a scripting language that allows compositions of web services and programming operations to accomplish business process goals. Web services may be composed into sequential and/or concurrent process flows. A BPEL process in turn is accessed as a web service via a WSDL interface. Thus, a BPEL process essentially becomes a broker, providing an arrangement, or orchestration, of other web services (including other BPEL processes). The programmability of BPEL includes operations like variable assignment, loop and conditional constructs, and fault and event handling. BPEL processes, like WSDL, are specified in XML.

BPEL possesses its own nomenclature for composition and components. A single unit of workflow processing is referred to as an activity (e.g., assigning values to variables, invoking a web service, etc.) Web services calls are known as invocations. Data are passed to and from web services through messages. Web services that are part of a BPEL process are thought of as partners and the references describing the accessibility of web services are referred to as partner links. The context of a service within a BPEL process is described as a partner role, which is actually just a semantic label for an endpoint reference, describing where a web service is located and how to access it. Activities that may be accomplished in parallel are encapsulated within a flow construct.

Several tools and models currently exist to abstract and simplify the implementation of BPEL. These tools significantly aid and accelerate process implementation and modeling is necessary to verify process consistency. However, the process developer must still possess a fundamental understanding of BPEL, its terminology, and its limitations. This is especially important if one wishes to implement high assurance BPEL processes.

BPEL addresses the high assurance concern of availability in two ways. Firstly, BPEL allows the dynamic changing of endpoint location (i.e., the location of web services to be invoked). This allows a BPEL process to use an alternate web service in the event that a previous one cannot be located or does not respond in a timely fashion. BPEL's second method of addressing availability is a byproduct of its nature as an internet-accessible service. Load balancers can be implemented to

balance web service requests for high-demand web services among many different servers providing the requested service.

Security is not an inherent part of BPEL. In order to implement a secure BPEL process, the process must be built on top of other rugged security mechanisms. Transport-level security may be achieved using the Secure Socket Layer (SSL), a widely used point-to-point security mechanism. To better protect integrity and confidentiality, message-level security mechanisms should also be employed. Web Services Security (WS-Security) is such a security mechanism that packages authentication information into each message to strengthen trust between web service consumers, BPEL processes, and remote web services.

1.2 Motivation

While BPEL is a useful and powerful scripting language for creating compositions of web services, its support for high assurance service-oriented computing is lacking. Some of BPEL's problems that complicate high assurance satisfaction are:

1. BPEL has a unique nomenclature that straddles the line between programmer and business process specialist. Fully understanding all of BPEL's terminology, capabilities, quirks, and deficiencies carries a significant learning curve for most people.
2. BPEL's power comes from its ability to compose complex orchestrations of an arbitrary number of web services into business process solutions. However, BPEL's powerful scripting ability also makes it quite easy for a process to succumb to logical errors and design inconsistencies.
3. BPEL is currently an evolving de-facto standard for web service composition. Processes created with BPEL today may not be compatible with the BPEL of tomorrow or they may not easily be able to exploit the latest advancements in BPEL and web service technology. There is also the distinct possibility that a different and incompatible web service composition language will replace BPEL in the future.

UML sequence diagrams model time-ordered interactions between entities. Interactions represent events and can express data traveling between entities. The entities may represent humans and/or non-human processes. We propose that UML sequence diagrams can suitably model BPEL processes and provide a remedy to these glaring problems. We demonstrate our proposal using two examples of BPEL processes that we have implemented.

2. Related Work

In addition to the business computing industry, BPEL and web services in general have drawn a fair amount of interest from the research community. This is due in part to the distributed processing benefit web services bring to computing. But the interest in web services is also due to the fact that web service technology is in its early infancy and there is considerable room for research and improvement.

O'Brien et al. discuss several quality attribute requirements that should be strongly considered when designing a software architecture that involves web services [1]. These attributes are interoperability, performance, security, reliability, availability, modifiability, testability, usability, and scalability. They also highlight the importance of acquiring suitable service level agreements to guarantee adequate satisfaction of these requirements from third party service providers. Kontogiannis et al. [2] identify three areas of challenges for adoption of service-oriented systems: business, engineering, and operations. They also reveal an underlying set of "cross-cutting" concerns that these areas share and propose a Service Strategy to address these concerns. Sarna-Starosta et al. [3] propose a means of achieving safe service-oriented architectures through the specification of service requirements using declarative contracts. Monitoring and enforcement of these contractual obligations are handled through the use of hierarchical containers and middleware.

Zheng et al. [4] propose a type of finite state machine, called Web Service Automata (WSA), to formally model web services, such as BPEL. They state that using WSA, they are able to model and analyze most of BPEL's features, including control flow and data flow. Their proposal includes a mapping from WSA to the input languages of the NuSMV and SPIN model checkers. Zheng et al. [5] use the WSA mapping to generate test cases in the NuSMV and SPIN model checkers. State, transition, and du-path test coverage criteria are expressed in Linear Temporal Logic and Computation Tree Logic. The logical constructs are used to generate counterexamples, which then provide test cases. These test cases are used to verify a BPEL process' control and data conformance and WSDL interface conformance.

In [6], Ye et al. address inter-process inconsistency through the public visibility of atomicity specifications. They adapt the atomicity sphere to allow a service to provide publicly the necessary details of "compensability and retriability" while keeping its proprietary details private. Their technique for constructing the atomicity-equivalent public views from its privately held process information involves the use of a process algebra, which they describe and prove mathematically.

Based on Service Oriented Architecture (SOA), the Bus model is a kind of service model to integrate heterogeneous services. Li et al. [7] develop a formal model for services, which has three levels: the programs model, the agents model, and the services model. The bus system is constructed from the parallel composition of the formal models. To exchange information, the service interacts with the

bus space instead of the other services, so concurrency is described by the global space of the bus system.

Chu et al. [8] design an e-business system using an architecture-centric design. They combine Semantic Web technology and e-business modeling to construct and semantically describe a service-oriented e-business model. The architecture-centric system design follows a "divide-and-conquer" method of decomposing the goal and defining and validating the architecture. The semantics definition helps discover registered services automatically and automated verification of system reconfiguration. In this way, the business goal can be mapped into services.

Dun et al. [9] model BPEL processes as ServiceNets, a special class of Petri net. Their approach constructs a formal model through a transformation into an S-Logic representation using an enriched form of reduction rules. They are able to analyze correctness and detect errors of the BPEL process from the ServiceNet's "throughness". Laneve et al. [10] propose a web transaction calculus, webπ, which assists in the verification of the compensable property for web service technologies that utilize web transactions as their fundamental unit of work. A compensable web service is one that facilitates the undoing of work should the web service fail to complete successfully. webπ is an extension to π-calculus. Web service languages are translated into webπ where their transactional protocols may be analyzed.

Foster et al. describe and implement a model-based approach for verification of web service compositions [11][12]. Their tool translates UML sequence diagram scenarios describing web service compositions into Finite State Process (FSP) algebra. The FSP algebra is then used for equivalence trace verifications of the compositions. Their tool also directly translates BPEL4WS into FSP algebra. In [13], Foster et al. discuss a model-based approach using finite state machines to represent web service compositions. They semantically describe the web service processes to verify compatibility between the processes and that the composition satisfies the overall system specification. In [14], Foster et al. present a detailed procedure for translating web service compositions expressed in BPEL4WS into Finite State Process (FSP) notation. They describe BPEL4WS constructs in terms of FSP semantics and analyze the mapping of specific activities using Labelled Transition Systems.

Akkiraju et al. [15] propose a framework for supporting web service compositions that provides functions such as security, access control, business partner discovery and selection, service level agreement monitoring, and logging. They believe that their framework fills in several of the inherent high assurance gaps of web service composition languages. Fu et al. [16] construct a Web Service Analysis Tool (WSAT) for analysis and verification of web service compositions. Their tool creates an automata-based intermediate representation of the composition. Control flow in the intermediate representation is restricted by the use of "synchronizability" conditions and Linear Temporal Logic and the SPIN model checker are then used to verify the composition and check its properties. Nakajima et al. [17] investigate the modeling of the Web Services Flow Language (WSFL) and

the benefit of model checking the web service compositions for reliability. They conclude that their model checking experiments successfully detect faulty flow descriptions and can be expanded to accommodate alternative semantics.

3. BPEL Process Examples

We present two examples that are drawn from one of our largest reservoirs of real business experience: insurance quoting and tracking. These examples are not meant to demonstrate elegant or solid BPEL process design. Both examples include simple errors. We use these examples to illustrate how our approach to BPEL modeling and analysis using UML sequence diagrams reveals problems of inconsistency and design errors.

3.1 Example 1

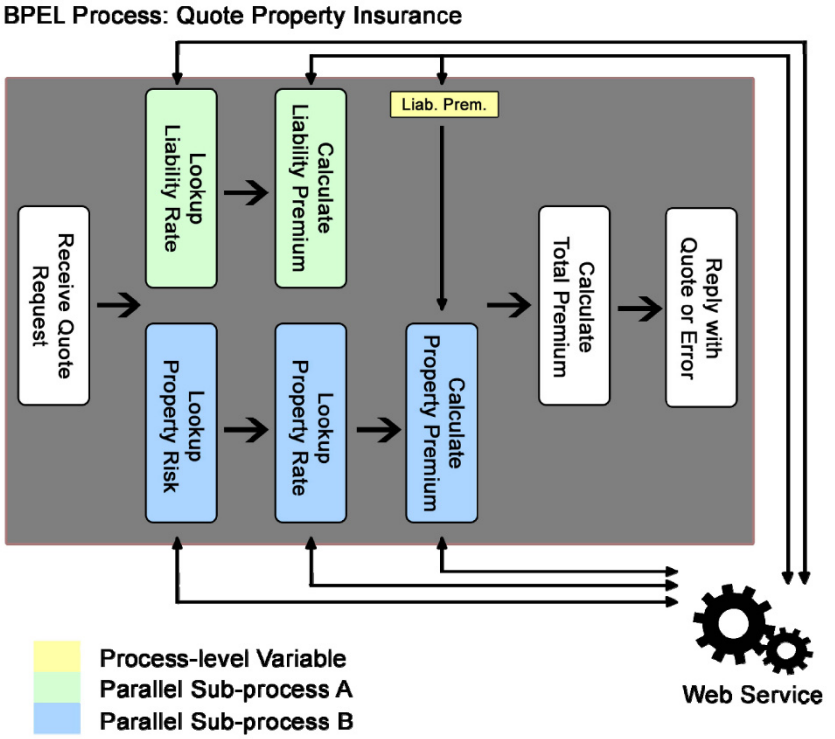

Fig. 2. The first example BPEL process for quoting insurance.

The first example of a BPEL process is an insurance quote processing and response system. The overall process design involves obtaining a quote for property and liability insurance (see Figure 2). The two different types of premium can be calculated independently using a Flow activity. There are several steps involved before a final quote can be determined (if at all) and returned to the BPEL service consumer. The BPEL process responsible for quoting property insurance proceeds through each step of the quoting process, calling other web services (located on either itself or remote servers). This example demonstrates an inconsistency error resulting from a mistake in implementation that creates a dependency between two concurrent sequences of activity in a BPEL flow activity.

The process steps of the first example are:

1. The first step of the example receives the request for the quote along with any data required for quoting (input requirements are specified within the WSDL document for the BPEL web service).
2. The second step of the process forks into two separate and concurrently executing sub-processes. Sub-process A requests a liability rate from the remote web service. Sub-process B requests property risk information from the remote web service.
3. Sub-process A requests liability premium to be calculated. The resulting liability premium is stored in a process variable. The Sub-process B requests a property rate from the remote web service.
4. Sub-process B requests property premium to be calculated, passing the previously calculated liability premium to determine if a discount modifier is necessary.
5. The process waits for both sub-processes to complete before requesting the remote web service to add the premium data and apply additional taxes and fees.
6. Lastly, the BPEL process replies to the calling service consumer, providing the resulting quote. If a fault occurs at any point during the BPEL process, the BPEL process replies to the calling service consumer with error information.

Below is simplified BPEL code for the first example, beginning with the flow activity, splitting the process into concurrent sub-processes:

```
<flow>
<sequence>
<invoke partnerLink="QuotePartner"
        operation="getLiabilityRate"
        inputVariable="propertyType"
        outputVariable="liabilityRate" />
<assign>
        <copy>
          <from variable="liabilityRate" />
          <to variable="liabilityPremiumInput" part="rate" />
        </copy>
```

```
        <copy>
          <from variable="insuredValue" />
          <to variable="liabilityPremiumInput" part="insuredValue" />
        </copy>
</assign>
<invoke partnerLink="QuotePartner"
        operation="calculateLiabilityPremium"
        inputVariable="liabilityPremiumInput"
        outputVariable="liabilityPremium" />
</sequence>
<sequence>
<invoke partnerLink="QuotePartner"
        operation="getPropertyRisk"
        inputVariable="zipCode"
        outputVariable="propertyRisk" />
<invoke partnerLink="QuotePartner"
        operation="getPropertyRate"
        inputVariable="propertyRisk"
        outputVariable="propertyRate" />
<assign>
        <copy>
          <from variable="propertyRate" />
           <to variable="propertyPremiumInput" part="rate" />
        </copy>
        <copy>
          <from variable="insuredValue" />
          <to variable="propertyPremiumInput" part="insuredValue" />
        </copy>
        <copy>
          <from variable="liabilityPremium" />
          <to variable="propertyPremiumInput" part="liabilityPremium" />
        </copy>
</assign>
<invoke partnerLink="QuotePartner"
        operation="calculatePropertyPremium"
        inputVariable="propertyPremiumInput"
        outputVariable="propertyPremium" />
</sequence>
</flow>
<assign>
        <copy>
          <from variable="propertyPremium" />
          <to variable="totalPremiumInput" part="propertyPremium" />
        </copy>
```

```
        <copy>
          <from variable="liabilityPremium" />
          <to variable="totalPremiumInput" part="liabilityPremium" />
        </copy>
</assign>
<invoke partnerLink="QuotePartner"
        operation="calculateTotalPremium"
        inputVariable="totalPremiumInput"
        outputVariable="quoteWithTaxes" />
<reply partnerLink="RequestorPartner"
        variable="quoteWithTaxes"
</reply>
```

3.2 Example 2

Our second BPEL process example also involves an insurance quote processing and response system (see Figure 3). When the process receives a request for a quote, it attempts to calculate premium based on parameters provided in the request. If the premium calculation succeeds, then a quote is emailed to the requestor and a quote request notification is emailed to the company's agents so that the agents may contact the requestor if the requestor fails to submit an application for insurance within a certain timeframe.

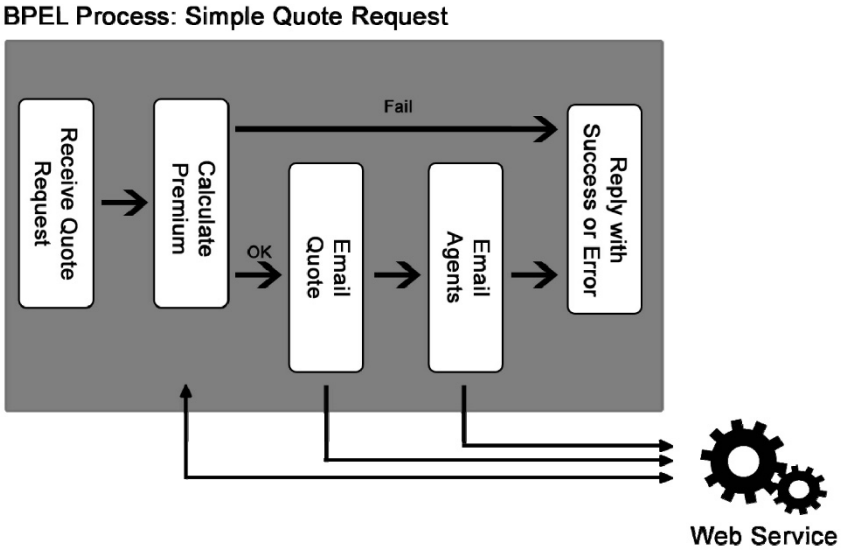

Fig. 3. The second example BPEL process for quoting insurance.

If an error occurs during the process execution, it is still important that email be dispatched to both the requestor and the agents. The requestor should receive a nicely formatted apology from the quoting system, along with alternate methods of contacting the insurance agency directly for a quote. Additionally, the agents need to know how to contact the requestor to try to assist them with the quoting process. Thus, the requirements for the second example state that the process should always email the requestor and the agents, regardless of the outcome of the process. Figure 3 clearly demonstrates that if the Calculate Premium activity fails then this requirement will not be satisfied. The second example serves to illustrate a discrepancy between the software requirements and the design and/or implementation of the software. We will show in the following section that our verification method can detect this discrepancy.

The process steps of the second example are:

1. The first step of the example receives the request for an insurance quote along with any data required for quoting (input requirements are specified within the WSDL document for the BPEL web service).
2. The BPEL process attempts to calculate premium based on the received parameters.
3. If the premium calculation invocation fails, the BPEL process replies to the requestor with an error and a "Sorry for the inconvenience" message. If the premium calculation succeeds, a quote is emailed to the requestor.
4. Company agents are notified of the request for a quote, along with the calculated premium. Our requirements dictate that this step should always occur, but our implementation fails to completely satisfy this requirement.
5. Lastly, the BPEL process replies to the calling service consumer with an indication of success or failure of the entire process.

These two examples are based on our real experience in the software engineering realm of the insurance industry. Both examples include simple errors that are typical of rushed or incomplete design. Detection of these errors using UML sequence diagram analysis will be demonstrated in the following sections.

4. Modeling BPEL Processes with Sequence Diagrams

The Unified Modeling Language (UML) provides a collection of modeling notations for describing different aspects of a software system, such as use-case diagrams, sequence diagrams, class diagrams, and state machines for requirements analysis and design. The sequence diagram is a key notation of UML to capture the interaction between the user, the system, and other components. A sequence diagram provides a scenario of one use case diagram using an intuitive graphical representation. Multiple sequence diagrams can be combined together to provide

the design of system. We model BPEL processes with sequence diagrams which will ease efforts in utilizing BPEL and enable them to detect hidden errors.

BPEL has structured activities that provide the execution order in a collection of activities. For example: the flow activity in BPEL represents concurrency and synchronization of multiple activities; pick represents nondeterministic choice from multiple activities, and so on. UML 2 provides some structured control constructs, such as combined fragments and interaction use, to express concurrent message exchange.

A sequence diagram has two dimensions, the vertical dimension represents time and the horizontal dimension represents objects participating in the sequence diagram [18]. In a sequence diagram, the vertical dash lines represent participants, called lifelines. The name of each lifeline is shown in the rectangle on the top of each dash line. The horizontal lines between lifelines represent messages passing between participants. The intersection points between lifelines and messages are called occurrence specifications [19].

Combined fragments, introduced in UML 2, represent different types of control flow. A combined fragment is composed by one interaction operator and one or more interaction operands. In Figure 4, the interaction operator “par” represents a parallel combined fragment, which has at least two interaction operands. In a parallel combined fragment, the occurrence specifications in the same operand keep their order but the occurrence specifications in different operands may execute in any order [19]. The negative combined fragment with the interaction operator “neg” has one interaction operand and it is not enclosed in other sequence diagrams. All the possible traces generated by this fragment are invalid traces [19]. The interaction operator “assert” represents the combined fragment as a mandatory behavior at that point in the sequence diagram. If the execution reaches the beginning of the assertion fragment, then the assertion fragment must execute. All other continuations result in invalid traces [18].

With these features, a UML sequence diagram can represent most BPEL structured activities, e.g., BPEL consumer, BPEL process, and web service are presented as lifelines, flow can be shown by a parallel combined fragment, and pick can be shown by an alternative combined fragment. Table 1 shows the mapping from BPEL constructs to UML sequence diagrams.

Modeling BPEL with sequence diagrams enables the building of tools to detect potential errors of system design. The number of errors needs to be minimized at the design level as this greatly helps to simplify the work of implementation and verification.

Table 1. Mapping from BPEL to UML sequence diagram constructs.

BPEL Activity	Activity Description	UML Sequence Diagram Construct	Construct Description
receive	wait for an incoming message to arrive	receive a message	the message can be synchronous or asynchronous
reply	send a message in response to previously received message	send a reply message	the message is synchronous
invoke	call a one-way or request/response operation (e.g., another web service)	send a call mess age	the message can be synchronous or asynchronous
assign	modify the value(s) of one or more variables	a reply message contains attribute assignments as arguments	the message is synchronous
throw	create a fault	send a fault message to the fault handler actor	fault handler actor is represented as a lifeline
exit	immediately end the BPEL process	combined fragment--break	the condition of the operand is true
wait	pause for a period of time or until a specified time	the timer actor sets a period of time	when a period of time elapses, a timeout message is generated
empty	do nothing (i.e., a no-op)	the timer actor sets one cycle	the actor generates a timeout message
sequence	perform enclosed activities in sequential order	combined fragment--weak sequencing	the messages execute sequentially
if	perform an activity based on condition satisfaction	combined fragment--alternatives or option	the conditions of all operands are mutually exclusive in alternatives
while	perform the enclosed activity as long as the condition is true	combined fragment--loop	the condition in BPEL is mapped to the condition of loop, minint=0, maxint=infinite
repeatUntil	perform the enclosed activity until the condition is true	combined fragment--loop	the negation of condition in BPEL is mapped to the condition of loop, minint=1, maxint=infinite
forEach	perform the enclosed activity a specified number of times	combined fragment--loop	condition=true, minint=0, maxint=N+1
pick	wait for one of many possible messages to arrive or a timeout	combined fragment--alternatives	at most one operand is chosen
flow	perform the enclosed activities concurrently	combined fragment--parallel	the messages in different operands are interleaved

5. BPEL Inconsistency Analysis

Different BPEL scenarios are based on different views, but they may be relevant to each other and conflicts that are called inconsistencies may exist among them. These inconsistencies can be detected by comparing multiple sequence diagrams with pre-defined inconsistency rules. Once detected, software engineers can fix the system design to remove the conflicts to make the software system consistent. To demonstrate detection of inconsistency in our examples, we provide two sample inconsistency rules.

Inconsistency Rule 1: detecting inconsistency between valid traces and invalid traces. Valid traces are generated from sequence diagrams with no negative combined fragments. Sequence diagrams with negative combined fragments generate invalid traces. An inconsistency exists when a valid trace associates directly with an invalid trace for predetermined properties of the software.

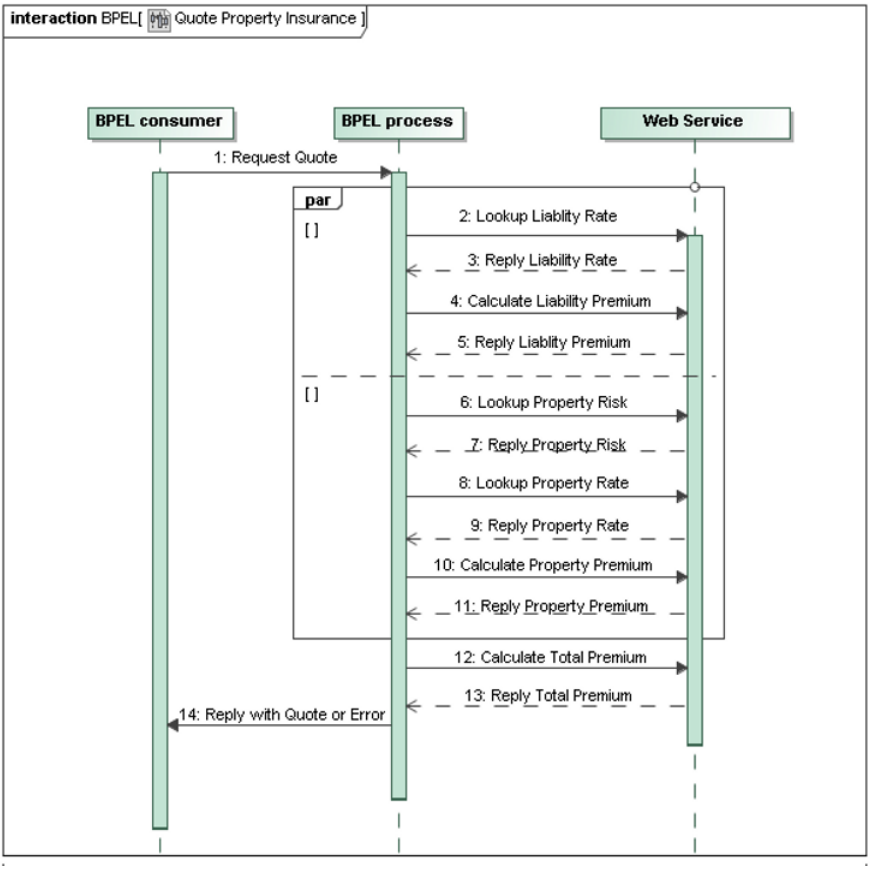

Fig. 4. BPEL Process: Quote Property Insurance.

Inconsistency Rule 2: detecting inconsistency between sequence diagrams with assertion combined fragments and other sequence diagrams. If any trace in a sequence diagram exists without the assertion and conflicts with a trace containing the assertion, then there is an inconsistency.

5.1 Analysis of Example 1

Figure 4 is a sequence diagram representing our first example of a BPEL process for quoting property insurance. When the BPEL process receives a quote request from a consumer, the process forks into two interleaving sub-processes and a task from either sub-process can be chosen to execute. One sub-process is for liability premium and the other is for determining a property rate. The interleaving relation of sub-processes is shown with a parallel combined fragment in the sequence diagram. The tasks in the same operand in Figure 4 keep their order, but tasks in different operands can be executed in any combination of orders. In this way, one sequence diagram can represent multiple execution traces.

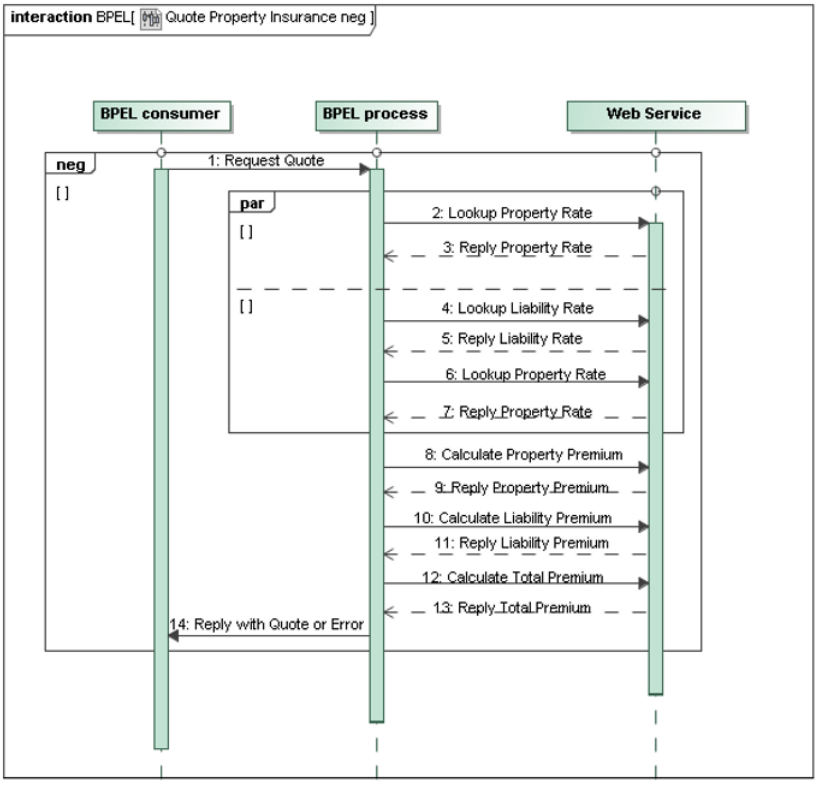

Fig. 5. One constraint in Quote Property Insurance.

This begs the question: are all of these execution traces valid? Is there a dependency among tasks from different sub-processes?

Assume that a software requirement states that the Calculate Property Premium task needs the value of Liability Premium in order to determine a discount modifier. It is easy to find the dependency that the Calculate Property Premium task should not happen before the Calculate Liability Premium task. Figure 5 shows this constraint in a sequence diagram with a negative combined fragment. Negative combined fragments define that all possible execution traces inside are invalid. A negative combined fragment tells a software engineer that the execution traces within the fragment should not happen in the software system. In Figure 5, the tasks inside the parallel fragment are still interleaving to each other, but the Calculate Property Premium task happens before the Calculate Liability Premium task. The negative fragments demonstrate that all the executions traces cannot happen. Comparing Figures 4 and 5, we detect an inconsistency and the design of system in Figure 4 will not provide a reliable implementation. Therefore, the design should be modified and the process re-verified with sequence diagram modeling.

5.2 Analysis of Example 2

Figure 6 is a sequence diagram of our second example BPEL process: Simple Quote Request. When a service consumer sends a request for an insurance quote to the Simple Quote Request BPEL process, the process invokes an external web service to calculate premium and the external web service replies with the resulting premium. The reply may be a success message (the resulting premium) or a failure message (with a specific error). After the BPEL process receives the reply message from the external web service, the process must email a quote to the consumer and inform the company agents of the request for a quote. Finally, the BPEL process replies to the BPEL consumer with a success or error message.

Software engineers may provide an execution trace of the system in Figure 7. In this execution trace, the BPEL process invokes the calculate premium operation from the external web service and receives a failure response. The BPEL process replies with a failure message to the BPEL consumer. The assertion fragment in Figure 6 is skipped. The assertion fragment in Figure 6 defines that after the BPEL process receives a reply from the external web service after calculating premium, only the Email Quote activity can happen. Instead, the reply from the BPEL process to the BPEL consumer happens in Figure 7. An inconsistency is therefore detected between the two sequence diagrams.

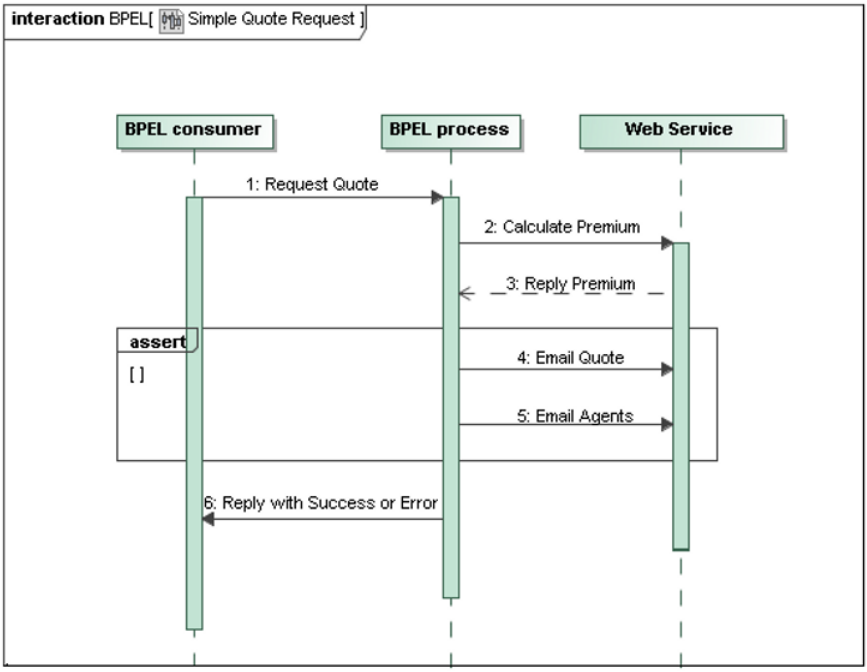

Fig. 6. BPEL Process: Simple Quote Request

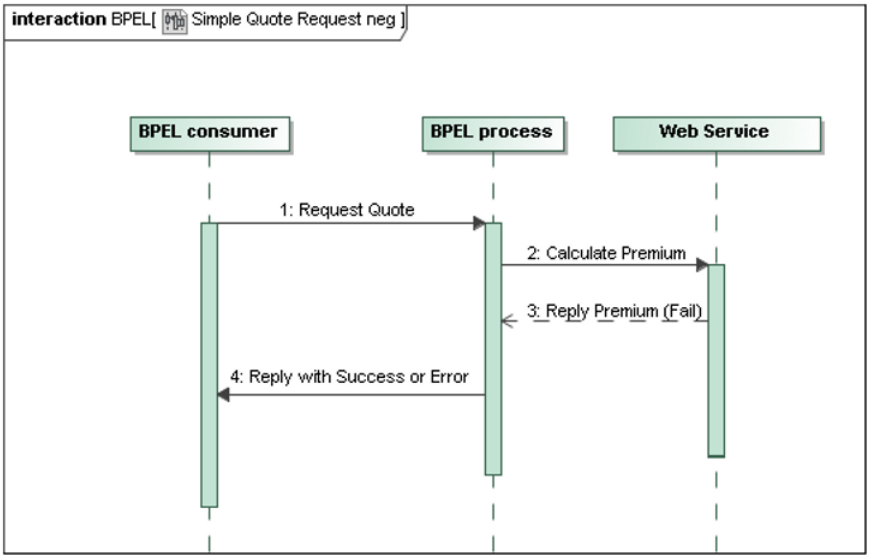

Fig. 7. One conflict in Simple Quote Request.

5.3 Evaluation

Our analysis technique has only two manual steps for discovery of inconsistencies. The first step is to create an initial sequence diagram in MagicDraw that models a BPEL process. The second step is the selection of inconsistency rules to use in trace generation. Once these steps are performed, traces are generated automatically, the inconsistency rules are applied, and the analyst is presented with a list of inconsistency warnings. The analyst must then examine each warning in the report and determine if the warning necessitates a design change.

By generating a negative combined fragment, our analysis method exposed an inconsistency in the implementation of our first example. The presence of traces within a negative combined fragment provides the software engineer with immediate knowledge of inconsistencies within the BPEL process. For our second example, we generated a valid execution trace for the BPEL process that did not include a required assertion combined fragment. This demonstrates a second type of inconsistency where the requirements of the BPEL process are not met (the activities of emailing the quote and agents)

Our inconsistency analysis approach with sequence diagrams facilitates rapid and thorough detection of inconsistencies within BPEL processes. Our approach may also be easily extended to accommodate other web service composition languages. Sequence diagrams are very intuitive, promote swift analysis, and inconsistencies between them (as traces) tend to be visually prominent. Additionally, there are currently many tools available to easily assist one in the generation of sequence diagrams.

6. Conclusions

Currently, BPEL is in a nascent state. It is a technology that straddles the line between software development and business process specification. As a result, BPEL contains some of the arcane expression of a programming language mixed with business-oriented terminology and process logic. BPEL is certainly a powerful abstraction language that can render compositions of distinct web services to solve business problems. These compositions can then be exposed as their own web services, which may be used by themselves in web service consumer applications or other BPEL processes. However, BPEL nomenclature carries a significant overhead for its initiates. Also, BPEL is not structured well to easily detect neither logical errors nor inconsistencies. Lastly, as an evolving de-facto standard for web service composition, what works for BPEL today may not work tomorrow.

For these reasons, we feel that BPEL implementation should be accomplished using abstracted design tools to simplify construction and ease the learning curve of BPEL's nomenclature. This will accelerate implementation and help reduce er-

rors. We have shown that modeling techniques such as our UML sequence diagram analysis approach can rapidly and automatically facilitate discovery of BPEL design flaws of inconsistency. Sequence diagrams are very intuitive and show temporal-based execution naturally. But tools and models do not obviate the need for the process designer to fully understand the fundamentals, quirks, and shortcomings of BPEL and web services. Such understanding is crucial in order to construct BPEL processes that hope to satisfy the different aspects of high assurance service-oriented computing.

6.1 Future Work

We feel that UML sequence diagrams hold some promise of a straightforward, well-adopted, and useful means of verifying consistency and reliability in BPEL processes. There is, of course, more investigation needed in this area. We also want to experiment with the use of UML sequence diagrams as a high-level BPEL process composition tool, generating BPEL code underneath the sequence diagrams.

The integration of web services within a software development project is an important consideration for software engineers. Web services may save time and money in implementation, simplify maintenance challenges, and enrich the overall specifications of the software. Currently, software engineers and workflow specialists must peruse registries of available web services and manually determine the suitability of each available web service in terms of functionality, cost, and interface specifications. There has been some research in semantically describing and automatically identifying web services within compositions and their initial results are promising. If web services can provide information regarding their specifications and context in a formal and standardized fashion, then the suitability of web services for a given software project could be determined automatically. This would allow engineers to simply "point" to a set of web service registries and receive a suitability report of all appropriate web services. The suitability report could then be automatically matched to the software project's own set of specifications to determine which implementation gaps could be satisfied by which web services.

State machines synthesized from our sequence diagrams may be adapted to provide BPEL process design. Whittle and Schumann [20] presented an algorithm for synthesizing state machines from multiple UML 1 sequence diagrams, which do not support structured control constructs. Uchitel et al. [21] provide a method to synthesize behavior models from multiple message sequence charts. Message sequence charts are similar to UML sequence diagrams. We may be able to synthesize state machine behavior models from UML 2 sequence diagrams such that we will be able to perform some formal analysis, like model checking.

References

[1] O'Brien L, Merson P, Bass L (2007) Quality Attributes for Service-Oriented Architectures. International Workshop on Systems Development in SOA Environments

[2] Kontogiannis K, Lewis GA, Smith DB et al (2007) The Landscape of Service-Oriented Systems: A Research Perspective. International Workshop on Systems Development in SOA Environments

[3] Sarna-Starosta B, Stirewalt REK, Dillon LK (2007) Contracts and Middleware for Safe SOA Applications. International Workshop on Systems Development in SOA Environments

[4] Zheng Y, Krause P (2007) Automata Semantics and Analysis of BPEL. Digital EcoSystems and Technologies Conference 147-152

[5] Zheng Y, Zhou J, Krause P (2007) A Model Checking based Test Case Generation Framework for Web Services. Fourth International Conference on Information Technology 715-722

[6] Ye C, Cheung SC, Chan WK (2006) Publishing and composition of atomicity-equivalent services for B2B collaboration. Proceedings of the 28th international Conference on Software Engineering 351-360

[7] Li Q, Zhu H, He J (2008) Towards the Service Composition Through Buses. High Assurance Systems Engineering Symposium 441-444

[8] Chu W, Qian D (2008) Architecture Centric System Design for Supporting Reconfiguration of Service Oriented Systems. High Assurance Systems Engineering Symposium 414-423

[9] Dun H, Xu H, Wang L (2008) Transformation of BPEL Processes to Petri Nets. Theoretical Aspects of Software Engineering 166-173

[10] Laneve C, Zavattaro G (2005) Foundations of web transactions. Proceedings of Foundations of Software Science and Computation Structures 282-298

[11] Foster H, Uchitel S, Magee J et al (2006) LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography. International Conference on Software Engineering 771-774

[12] Foster H, Uchitel S, Magee J et al (2006) Model-based Verification of Web Service Compositions. 18th IEEE International Conference on Automated Software Engineering

[13] Foster H, Uchitel S, Magee J et al (2004) Compatibility Verification for Web Service Choreography. 3rd IEEE International Conference on Web Services

[14] Foster H (2003) Mapping BPEL4WS to FSP, Technical Report. Imperial College

[15] Akkiraju R, Flaxer D, Chang H et al (2001) A Framework for Facilitating Dynamic e-Business Via Web Services. OOPSLA 2001 - Workshop on Object-Oriented Web Services

[16] Fu X, Bultan T, Su J (2004) WSAT: A tool for Formal Analysis of Web Services. 16th International Conference on Computer Aided Verification

[17] Nakajima S (2002) Model-Checking Verification for Reliable Web Service. Workshop on Object-Oriented Web Services

[18] Rumbaugh J, Jacobon I, Booch G (2004) The Unified Modeling Laguage Reference Manual Second Edition. Addison-Wesley, United States

[19] Object Management Group (2007) Unified Modeling Language: Super-structure v2.1.2.

[20] Whittle J, Schumann J (2000) Generating statechart designs from scenarios. International Conference on Software Engineering 314-323

[21] Uchitel S, Kramer J, Maggee J (2003) Synthesis of behavioral models from scenarios. IEEE Transactions on Software Engineering 99-115
[22] Arlow J, Neustadt I (2008) UML 2 and the Unified Process, Second Edition. Addison-Wesley, United States
[23] OASIS (2007) Web Services Business Process Execution Language Version 2.0.