International Journal of Computer and Communication Technology

Volume 5 | Issue 1 Article 10

January 2014

Event Based Approach for Web Services Composition

A. Vani Vathsala
CVR College of Engineering, JNTU, Hyderabad, India, atlurivv@yahoo.com

Hrushikesha Mohanty
Department of Computer and Information Sciences, University of Hyderabad, Hyderabad, India.,
mohanty.hcu@gmail.com

Event Based Approach for Web Services Composition

A.Vani Vathsala, Hrushikesha Mohanty
CVR College of Engineering, University of Hyderabad
JNTU, Hyderabad, India
atlurivv@yahoo.com, mohanty.hcu@gmail.com

Abstract- The success of the Internet and the ongoing globalization led to a demand for new solutions to meet the requirements for IT-systems. The paradigm of service-oriented and event-driven architecture with fine grained and loosely coupled services tries to cope with those needs. Service Oriented Architecture (SOA) and Event Driven Architecture (EDA) are two acknowledged architectures for the development of business applications and information systems, which have evolved separately over the years. Today both architectures are acknowledged, but their synergy is not. There are numerous benefits of having an architecture that supports coexistence between operations and events, and composition of services based on operation invocation and event triggering. As part of our ongoing research work, we have tried to analyze in this paper, the basic design of Event based systems, issues that have to be addressed when event based approach is used for composing and coordinating web services. Then we have specified the techniques available that handle these issues, and gave a comparative study on these techniques. Finally we have attempted to sort out the unhandled/ partially handled issues that could be addressed as part of our research.

I. INTRODUCTION

Web services (WS) is an evolution in the way of architecting, designing, implementing and deploying internet based e-business solutions. WS, based on simple architectural concepts, focuses on the interactions of components on the network, not on the details of what happens within an individual component (service).

To capture real Business to Business (B2B) or Business to Consumer (B2C) interactions, a set of services need to work together and be executed in specified order. Such execution is termed service composition or choreography. Several standards have been proposed and are being introduced into practice, especially BPML (Business Process Modeling Language), BPEL4WS (Business Process Execution Language for Web Services), and WSCI (Web Services Choreography Interface). In this context, formal methods can contribute to increased reliability and consistency in the web services composition design process.

The use of the event-based paradigm for application integration is not really new. However, the major part of research on event-based architectures is devoted to the design of notification servers and the development of event-service middleware. Little attention is paid to composition of web services, specifically in event-based service oriented architectures. There are plenty of approaches to formalize Web Service composition: Relation Algebra, Pi Calculus, and Petri nets etc.

The problem is to identify the issues to be handled when event driven approach is used for web services composition. Then study various formal methods available to formalize Event Driven Service Oriented Architectures. These techniques are studied with an idea to analyze their approach and short comings, so that they can be taken care of in our proposed research.

This paper is structured as follows: In section 2 basics features of event based systems are discussed. I n section 3 we have discussed the need for Event based Web Services and the issues to be handled while designing these systems. We have studied few papers on Event based Web services composition in section 4 giving their merits and demerits. In section 4 we have presented our proposed approach for web services composition in event driven systems. We conclude with an outlook on ongoing research work.

II. EVENT BASED SYSTEMS

Event based systems (EBS) have received increasing attention in the past decade from various communities. Central to these systems is the notion of event, which is often considered as “a Happening of interest”. Building applications and systems around an event-driven architecture allows these applications and systems to be constructed in a manner that facilitates more responsiveness, since event-driven systems are, by design, more normalized to unpredictable and asynchronous environments

An event-driven system is typically comprised of event consumers and event producers. Event consumers are also called as Event sinks or Event processing engines. Event sources are also called as Event Producers or Event Emitters. Event consumers subscribe to an intermediary event manager, and event producers publish to this manager. When the event manager receives an event from a producer, the manager forwards the event to the consumer. If the consumer is unavailable, the manager can store the event and try to forward it later.

Basically, there can be two types of events:

Business events: Events may be constructed in a component system to indicate the happening of a business event (for example arrival of stock in inventory management system). When such an event occurs, a component can

inform the other components by explicitly generating an event message and sending it to event manager. Each business event has a name and a list of arguments. The event (with its arguments) is received by one or more consumer of the event.
Component-triggered events: Events can be defined for *before, after, instead of,* etc. the execution of a component. The rules can be used to customize the behavior of the component and/or to satisfy some enterprise-wide integrity constraints or business policies.

ISSUES

2.1 Event Specification: Events exchanged between and among applications in complex information systems allow the various facets of the system to inter operate, communicate and coordinate their activities. Event definition has to ensure completeness of the data by providing properties to publish the following information:

- The identification of the event given by globally unique *event id.*
- The identification of the component that is reporting the event, *Component Id*
- The type of the event, business event or method triggered event.
- The *timestamp* of the event
- The identification of the component that is affected by the situation (which might be the same as the component that is reporting the situation)
- The situation itself

An event may consist of two parts: the header and the body. The information presented above can become a part of event header. Event attributes can be used to represent this information. The event body contains the user generated payload of the event. The event body describes what has happened.
If low inventory threshold event is published by a component, the event body would contain the information to communicate which product fell below the minimum threshold.. For the low inventory threshold event, the event body would contain the product identifier, the product description, timestamp of the event, and threshold levels.

Preconditions or filters represent an obligation for those event sinks that wish to trigger the execution of an event. These filters are expressed using predicates on event attributes. Although testing the preconditions is in principle the responsibility of the event sink, one can also develop architectures where the testing of the preconditions resides within the event manager or even within the event source.

2.2 Event Publication: An event publication is an instance of an event type that has a type schema that describes the event type. This links the concept of a programming language type to an event. An event type contains an event type name and defines a set of event attributes, which are typed attribute/value pairs that hold the data in an event instance of this event type. Event publications can thus be regarded as record-like structures. An event type may also be associated with a parent event type, inheriting all its event attributes and entering into a sub typing relation with that type and all its ancestor types.

2.3 Event Subscriptions: By the same reasoning as for publications, event subscriptions can be typed as well. Event subscribers can express their interest in events in two stages. First, an event type (or a set of event types according to the event type inheritance relation) is chosen by the subscription. Then, a filtering expression is provided that selects events of the specified event types that satisfy a set of predicates over their event attributes.

2.4 Event Notification: The purpose of the Event Notification subsystem is to notify event sinks of events that occur. Event sinks subscribe to the notification subsystem and can specify the events for which they want to receive notifications. Event notifications are messages that are sent by the event manager to all event sinks who have subscribed for that particular event.

2.5 Event Monitoring: Event consumers subscribe to an intermediary event manager, and event producers publish to this manager. When the event manager receives an event from a producer, the manager forwards the event to the consumer.The simplest architecture is to use a central Monitor that relays all event messages. Publishers push events to this Monitor component and subscribers receive events from the same component. However, while this architecture is simple, the central component yields a bottleneck for scalability. Alternative architectures therefore avoid a central component by using network of so called brokers. Such a decentralized architecture also enables to improve reliability.

2.6 Event Handling: Every event is associated with an action that will be performed when the event occurs. The associated action can be logging the event, routing the event to the electronic address of a user, or sending the event to an application program for further processing

etc.In some approaches, when multiple consumers are notified of an event occurrence, it is assumed that these consumers will act independently from each other and that no response to the event producer is required. In other approaches the event producer can receive feedback on the execution status of the coordinated action associated with the triggered event. Moreover, in addition to the possibility of providing the event producer with a return value, the responses of the event consumers need to be coordinated (either all consumers execute the corresponding action successfully or all the action associated to the event are rolled back).

III. EVENT BASED APPROACH FOR WEB SERVICES

In the enterprise context, business events, e.g., a customer order, the arrival of a shipment, or the payment of a bill, may affect the normal course of a business process at any point in time. This implies that coordination among business processes cannot be designed apriori assuming that events are predetermined following a particular flow, but must be defined dynamically, driven by incoming, parallel and asynchronous event flows. Enterprise applications then must communicate using an event-driven SOA. An event driven SOA (ED-SOA) thus denotes an architectural approach on how enterprises could implement an SOA, respecting the highly volatile nature of business events.
Event Driven SOA supports one-to-many communication, rather than one-to-one communication among services in SOA. As there can be more than one event handler for handling single events, Event Driven SOA can support one-to-many communication.

Although ED-SOA results in complete decoupling of services, a service which acts as event sink must know certain details about the events that it would handle. These details would be used by event sinks to construct the filter expressions for precondition checking. These details about events are maintained by event manager and are referred to as **Event Data.** This Event Data should include list of all published events along with their headers and bodies.

Various ***issues*** to be addressed in Event Driven Web Services are:

- Service Specification
- Event Publishing by a service
- Event Subscription by a service
- Service Composition

In addition to Business events and method triggered events, ED-SOA variant requires one more kind of event: **Operation Invocation event.** Services in the ED-SOA variant are not required to understand protocol implementations or have any knowledge on routing of messages to other services. Hence to invoke another web service the service requester need not know the exact address of service provider. It just has to get the Event data from event manager and construct an operation invocation event by specifying the service required, and the parameter values. Once this event is received by event manager, it will be converted into operation invocation since it has the required details of service providers.

3.1 SERVICE SPECIFICATION

To combine SOA and EDA concepts, services can be extended with the notion of events [7]. Services, which expose operations through interfaces (port types), can be extended to act as event producers and event consumers. Events in the ED-SOA can use the XML representation of the associated data, which makes event structure flexible, maintainable, and easy to understand. Hence events can be represented as XML or textual messages that are exchanged between event sources and sinks via event manager.

Events in ED-SOA variant can be specified with the same Event header and body schema, where event header may consist of the following information:

- Event name
- Event Id
- Event type (business/method triggered/operation)
- Timestamp
- Valid from
- Valid till

In case of explicit business events and component-triggered events event body must include all the details which reflect the situation that has raised the event. These details can be provided by means of event attributes.
In case of Operation invocation event, the event body should contain details about

- Name of the Service to be invoked and the operation to be invoked.
- Parameter names and their values.

Any Service Provider has to provide the following details to the event manager:

- Service Name
- Service Id
- Description of the service provided
- Operations provided including signatures of the operations
- The port address at which service is available

Upon receiving Operation invocation event the event manager has to inspect the event body to determine which service has to be invoked. Then event manager must send

operation invocation request to the event sink available at the port address specified.

3.2 Event Publishing

Any communication between services in ED-SOA variant is through event manager. If any service wishes to act as event source, event manager has to be notified about it.

A Service which acts as event source has to publish the events that it will generate by sending publish-messages to the event manager. The details which have to be included in the publish-message are:

- Service Id
- List of Events that it will generate.
- For each event :
    - Event header
    - Contents of the event body
    - Valid from
    - Valid till

3.3 Event Subscription

A Service which acts as event sink has to subscribe to the events that it will handle by sending subscription-messages to the event manager. The details which have to be included in the subscription-message are:

- Service Name
- Service Id
- List of Events that it will handle.
- For each event :
    - Filter Expression that would be constructed using event attributes from body of the event.
    - Output Parameters if any
    - Subscription valid from
    - Subscription Valid till

3.4 Service Coordination

To achieve a more lightweight arrangement an event driven SOA requires that two participants in an event (server and client) be fully decoupled, not just loosely coupled. With fully decoupled exchanges the two participants in an event need not have any knowledge about each other, before engaging in some business transaction. In this case, there is no need for a service (WSDL) contract that explicates the behavior of a server to the client. The only relationship is indirect, through the event Manager, to which clients and servers are subscribed as subscribers and publishers of events.

The event-based broadcasting paradigm offers the additional advantage of a complete separation of the coordination aspects from the functionality aspects. The responsibility for the coordination resides completely within the event dispatcher, whereas the functionality part resides within the participating applications or components.

3.5 Service Composition

The event paradigm is in the first place a composition mechanism, which allows for the coordination of the respective web services that together form a more complex system [8]. In a service composition context, event-based interaction occurs at the level of web services. The service that triggers the event is an event producer; the services with subscriptions are event consumers

In order to efficiently tackle the service composition problem, we require a framework for discovery and selection of suitable services in the first place. The composition technique adopted has to compute functional matching of the services while establishing a sequence. In a typical event-driven SOA environment user requests are not evidently tasks. Instead, they have to be modeled as events and the service composition problem has to be handled as event processing.. Each event corresponds to some desired target event/s that has to be handled by a service composition process.

In the context of dynamic web service discovery, functional mismatch (incompatible sequence constraints) cannot be resolved by modifying the web service and one will continue to search for another web service or set of web services with compatible sequence constraints.

IV. EVENT BASED WEB SERVICES COMPOSITION: AVAILABLE APPROACHES

A. Graph based Service Composition

A proactive event-driven model is proposed in [12] where user activities and services are treated as events. SOA system is then graphically modeled into a network of activities called “Activity Network” (ANet). An ANet can be viewed as a workflow of events. However, ANet differs from most workflow models in the sense that it incorporates the functional and contextual information of services within itself. Hence, the workflow nodes in ANet are not simply processes but are event vectors (both services and user events). This sort of treatment allows us to perform a variety of operations over the network including causality reasoning and inter event compatibility computation. The nodes of such a network are the services and the edges their mutual causal dependency. It is then shown how both functional as well as contextual information (like list of entities getting affected by the event, state changes in those entities due to the occurrence of the event, action taken by those entities etc) can be incorporated into the network and then justified why that is necessary for solving the problem of service discovery, selection and composition.

As can be understood by the definition of ANet, with the addition of more and more services (especially functionally and contextually similar services) the number of dependency defined will be large. The ANet becomes more complex as it grows with time. Thus, service discovery, selection and composition become much more complicated due to the potential explosion of the search space. Performing the basic operations over the ANet becomes computationally costly. In order to resolve this issue efficiently an abstraction technique termed *ANet Abstraction is* proposed that transforms a potentially complex ANet into a simpler abstract ANet

Analysis **:** In[12] Problem of service discovery, selection and composition has been addressed using graph based technique instead of AI techniques. Although functional matching of services is taken care of, graph and tree traversal algorithms are costly when the number of nodes is more. Hence the performance of proposed approach may be affected with the number of nodes. The approach addresses business events, method triggered events and operation invocation events.

B. Business events as atomic contracts for component integration

Reference [8] proposes the concept of *business events* as the cornerstone to web service description and coordination. First, web service architecture is introduced as the result of an event based analysis & design phase. Then, it is advocated how the event concept can be used for semantically rich web service description. Furthermore, a web service *composition model* is proposed, based on *event broadcasting* and *event preconditions*, instead of traditional one-to-one method invocations. Event sequence constraints are modeled using State machines. Using state machine representation, the constraints are encoded into a component event table.

The coordination between components is achieved by having components specify preconditions for business events. As a result, a business event becomes a small scale contract between involved components: each component can insert its own clauses into the contract by specifying preconditions. Additionally, post conditions can be used to given a guarantee on the results. Obviously, this interaction pattern easily accounts for changes in the number of components that need to be notified of an event: it suffices to equip the event dispatcher with a subscription mechanism. Whenever a component needs to be notified of a particular type of business events, it subscribes with the appropriate event dispatcher to the corresponding event. When a component subscribes to an event, it means that the corresponding cell is marked in the component–event table. Three kinds of information are associated with such entry:

- The preconditions specify the conditions that need to be satisfied in order to accept the execution of the event.
- The actions specify the services or methods that must be called in order to notify the component of the occurrence of an event.
- The post conditions specify the guaranteed results.

When a component no longer needs to be notified of a particular type of event, it suffices to remove the subscription by removing the marked cell from the component–event table. A limitation is that the event sequence constraints only represent control dependencies between the component's activities. There is no direct construct for modeling data dependencies. An example of the latter could be the requirement that an invoice event is only allowed after some total_price attribute has been assigned a value.

Analysis: In the current incarnation, formalism is provided by using Process Algebra. The approach addresses business events, method triggered events and operation invocation events.

V. THE PROPOSED APPROACH

In order to formalize EDSOA, it is planned to follow the steps given below:

1. Develop an Abstraction on Event-driven SOA (Presented in section III)
2. Model EDSOA using Petri Nets
3. Develop programming primitives to implement the model
4. Tool that supports the proposed EDSOA to compose

- A web service
- A coarse-grained web service as a composition of many fine-grained services.
- Dynamically coarse-grained service.

The approach that we have selected for event driven Web service Composition is to use Petri nets. The Petri net formalism is a popular and powerful process modeling technique for the representation of processes which exhibit concurrency, parallelism, synchronization, and mutual exclusion.

Introduction to Petri Nets

A Petri net is also known as a place/transition net or P/T net. A Petri net is a bipartite graph in which nodes represent Transitions (rectangles) and Places (Circles) and edges form arcs. Places are usually used to model resources or states of a system. Transitions are used to model activities which change the values of conditions and resources. Arcs run from a place to a transition or vice versa, never between

places or between transitions. The places from which an arc runs to a transition are called the input places of the transition. The places to which an arc runs from a transition are called the output places of the transition.

Places may contain tokens. A token in a place can indicate whether a condition associated with a place is true or false. Tokens may be used to represent specific value of the condition or resource. A distribution of tokens over the places of a net is called a marking. Hence the state of the system may be modeled by marking the places with tokens. A transition of a Petri net may fire whenever there is at least one token at the start of all input arcs. When it fires, it consumes these tokens, and places tokens at the end of all output arcs. A firing is atomic, i.e., a single non-interruptible step.

Example

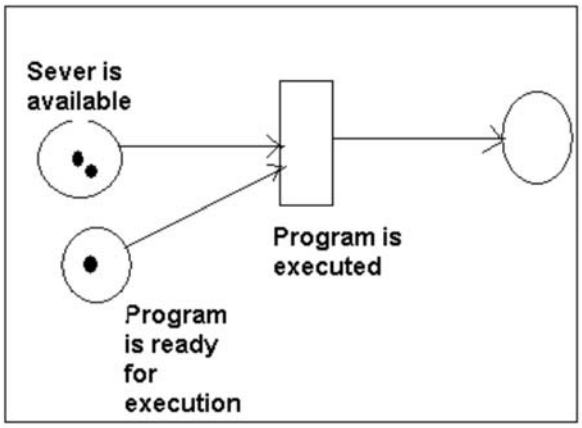

Fig 1: Initial Marking

Fig 2: After Firing

Using Petri Nets to model Event Driven (ED) Systems

Petri nets can model Event Based systems. They can capture events that occur in a sequence, that occur concurrently, mutually exclusive events etc. Each Place can represent an event type. Since events may have attributes, each place is defined along with these attributes. A token at a place represents an occurrence of the event. Each token is

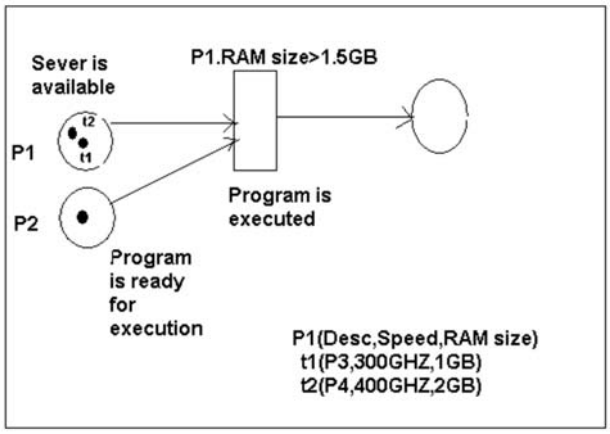

Fig 3: Modeling ED systems using Petri Nets

associated with exact attribute values. Execution of the action associated with the event is represented by firing of the transition. Each transition may be annotated with a filter

expression which has to be true for the transition to fire.

Modeling Event Driven Web Service Using Petri Nets

A Web service is viewed as a composition of events and actions. A Petri net can be used to model the behavior of a web service. Each arc may be annotated with a condition which results in the generation of the token

Example:-

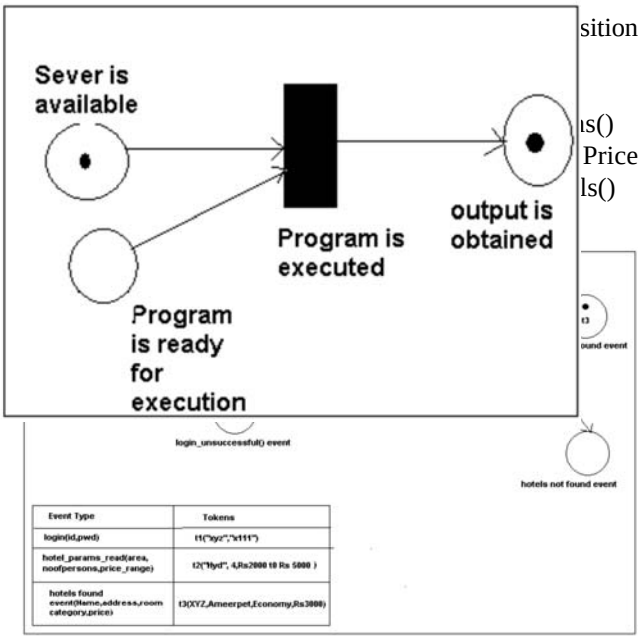

Fig 4: Modeling ED Web services using Petri Nets

Currently we are working on how to use petri nets to model event driven systems and perform functional and content based matching required for event based web service composition.

VI CONCLUSION

In this paper we have surveyed the basics of Event driven approach and issues to be handled when Web services are made event driven. Then we have reviewed few papers on Web service composition in event based systems, and analyzed their approach. Since Petri nets lend themselves naturally for representation of Web services we have proposed to use Petri nets for formalising Event driven Web service compositions quoting their advantages. As the research proceeds, we plan to model ED-SOA variant using petri nets, by adding additional features to petri nets. Then this model has
to be used to formalise composition of Web services.

REFERENCES

1. Peter H¨ofner1 Florian Lautenbacher2.Algebraic Structure of Web Services. Electronic Notes in Theoretical Computer Science 200 (2008) 171–187.
2. Roberto Lucchi, Manuel Mazzara. A pi-calculus based semantics for WS-BPEL. The Journal of Logic and Algebraic Programming 70 (2007) 96–118.
3. Faisal Abouzaid and John Mullins. A Calculus for Generation, Verification and Refinement of BPEL Specifications. Electronic Notes in Theoretical Computer Science 200 (2008) 43–65
4. Faisal Abouzaid and John Mullins. Formal Specification of Correlation in WS Orchestrations Using BP-calculus. Electronic Notes in Theoretical Computer Science 260 (2010) 3–24.
5. Rachid Hamadi Boualem Benatallah. A Petri Net-based Model for Web Service Composition. Fourteenth Australasian Database Conference (ADC2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, Vol. 17
6. Wei Tan, Yushun Fan, and MengChu Zhou. A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. IEEE transactions on automation science and engineering, vol. 6, no. 1, January 2009
7. Matjaz B. Juric .WSDL and BPEL Extensions for Event Driven Architecture. Information and Software Technology (2010), doi: 10.1016/j.infsof.2010.04.005
8. M. Snoeck *, W. Lemahieu, F. Goethals, G. Dedene, J. Vandenbulcke. Events as atomic contracts for component integration. Data & Knowledge Engineering 51 (2004) 81–107.
9. Olga Levina, Vladimir Stantchev.Realizing Event-Driven SOA. 2009 Fourth International Conference on Internet and Web Applications and Services.
10. Mike P. Papazoglou · Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal (2007) 16:389–415 DOI 10.1007/s00778-007-0044-3
11. Zakir Laliwala. Event-driven Service-oriented Architecture for Dynamic Composition of Web Services.Ph D Thesis submitted to Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar in 2007.
12. Sourish Dasgupta, Satish Bhat, Yugyung Lee. An Abstraction Framework for Service Composition in Event-driven SOA systems. 2009 IEEE International Conference on Web Services