Revisiting proactive service-oriented architecture: from design and implementation to validation and performance improvement

Pengcheng Xiong 1, Qingyang Wang 2, Liangjie Zhang 3, Calton Pu 4

1 College of Computing, Georgia Institute of Technology
266 Ferst Dr., Atlanta, GA 30332, USA
1, 2, 4 {pxiong3, qywang, calton}@cc.gatech.edu
3 Kingdee International Software Group Company Ltd
Shenzhen 518057, China
3 zhanglj@ieee.org

Abstract—Compared with the first version of service-oriented architecture (SOA) where clients invoke the web service by sending a request to the service provider and receiving a response when the service is done, the second version of SOA is enriched by events to possibly automate web service invocation following event patterns. In this paper, we design and implement proactive service-oriented architecture (PSOA), which monitors the happening of interested events and invokes desired web services when the events satisfy pre-defined triggering conditions according to SOA 2.0. The core system which is composed of event monitoring, triggering condition evaluation, and services invocation is implemented by an event filter manager, a triggering condition evaluator and a service invocation manager, respectively. Beyond the design and implementation, we make two main contributions with respect to the validation and performance improvement. (1) We use a formal method based on Petri nets to model and validate the correctness of the new PSOA system. (2) We evaluate the scalability of the system and propose a “grouping” based method to further improve the average response time as well as the throughput of the system under large-scale situation.

Keywords—Service-oriented architecture, Petri nets, system validation, system performance management.

1 INTRODUCTION

According to service-orient architecture (SOA) 1.0 [23], clients invoke the web service by sending a request to the service provider and receive a response when the service is done. This is a “pull” model of traditional request/response interactions. The main drawback of this kind of model is that the client has to be continuously busy in this activity until it receives the expected result. The disadvantage of SOA 1.0 becomes more obvious when web services monitor one or more event streams [19], [5], [18] and take actions when a set of predicates based on the events are satisfied. A typical example is a real-time financial web service: “if both IBM and HP stock prices fall by more than 5% within a 24-hour period, then sell 20% of the entire technology stock portfolio.” Another example is environment-sensitive web service, where severe weather (e.g., hurricane) or disaster (e.g., earthquake) would automatically trigger the movement of appropriate supplies (food, materials, tools, and/or appliances) towards the affected areas.

This motivates the service-orient architecture (SOA) 2.0 [2] where web service applies the idea of Event-Condition-Action (ECA) to take quick actions based on pre-defined set of predicates for some specific type of events. The idea of SOA 2.0 is similar to other related work, e.g., continual queries(CQ) [12], publish/subscribe system [9] and active databases [14]. For example, in continual query, we have an active database that defines triggering predicates using ECA rules. The predicates are used by the queries from the clients. Once the expected event happens, the condition is evaluated to be true according to the ECA rule and then the query results are pushed to the client. Similarly, we have an active web service by adding ECA rules in SOA 2.0: events are typically a stream of events; triggering conditions are predicates on the events that enable the execution of action; and actions are execution of services.

Following SOA 2.0, we have implement a proactive service-oriented architecture (PSOA) system as shown in Figure 1. The system consists of three main components.
The first component on the left is the event streams and event filter systems. Appropriate existing event processing and filtering systems for chosen applications can be linked into the PSOA system with the help of customized software wrappers. The second component in the middle is the triggering condition processing, which integrates external and contextual information into the evaluation of triggering conditions. The third component on the right is the SOA services (with appropriate wrappers) that will be executed as actions when the triggering conditions are evaluated to be true. The information flows in from the event filters and through to the execution of triggered SOA services, with the response(s) pushed to the interested users.

Beyond the design and implementation of proactive service-oriented architecture (PSOA) system, we focus on the validation and performance improvement. We are interested in answering two critical questions: (1) whether the proactive service-oriented architecture (PSOA) system is sound, i.e., whether the proactive service-oriented architecture (PSOA) system is able to invoke correct web services given initial input events streams; and (2) whether the proactive service-oriented architecture (PSOA) system is scalable to a large number of users/requests.

The main contribution of this paper can be summarized as below: (1) We propose a formal method based on Petri-nets to model and validate the correctness of the proactive service-oriented architecture (PSOA) system. (2) We propose a group-based approach to improve the performance of the system with respect to the response time and the throughput under large-scale situation.

The rest of the paper is organized as follows. Section 2 introduces proactive SOA event service and describes the details about design and implementation. Section 3 defines the soundness property of a PSOA system and discusses the guarantee of this property through Petri nets modeling techniques. Section 4 presents the evaluates the performance of

The prototype system through real experiments and proposes a group-based approach to improve the performance of the system. Section 5 summarizes the related work before Section 6 concludes the paper.

2 PSOA SYSTEM DESIGN AND IMPLEMENTATION

2.1 PSOA System overview

A PSOA system is built on top of continual queries(CQ) [12] to provide proactive services as shown in Figure 2. It is composed of five parts, an installation manager, a PSOA controller, a triggering condition evaluator, a service invocation manager and an event filter manager. The PSOA controller manages the remaining four parts. Generally, after a user defines a new PSOA, the installation manager generates the code corresponding to the PSOA's specification of event filtering, condition triggering and service invocation. The code is then deployed into the correspondent managers.

We will use the following concrete example throughout the paper. Assume that a user defines PSOA_0001 as “reserve a Delta airline ticket from Atlanta to New York City on 06/01/2012 if the weather on that day is sunny (the corresponding weather code is 60), a Delta flight costs less than $200, and the Hilton hotel in NYC (zip code is 10001) has a standard room available that costs less than $200.”. It is expressed as follows:

Create PSOA_0001 Airline_Ticket_Reservation as

Event_monitoring:
Weather update from weather.yahoo.com
Flight update from www.delta.com
Hotel update from www.priceline.com

Triggering_condition:
(Weather.zip = ‘10001’ ∧ Weather.date = 


```java
'2012-06-01' \ WEATHER\text{.code} \ '60' \}
\land
\{\text{Flight\_company} = 'Delta' \land \text{Flight\_depart\_city} = 'Atlanta' \land \text{Flight\_arrival\_city} = 'New York City' \land
\text{Flight\_depart\_date} = '06/01/2012' \land \text{Flight\_price} < 200\}
\land
\{\text{Hotel\_name} = 'Hilton' \land \text{Hotel\_location} = '10001' \land \text{Hotel\_price} < '200' \land \text{Hotel\_checkin\_date} = '2012-06-01'\}

\textbf{Stop condition:}
Once invoked or expire after one month

\textbf{Service invocation:}
Invoke booking\_ticket\_service

We can see that PSOA\_0001 could be triggered if all of the three events are satisfied, i.e., weather, flight and hotel. The event filter manager is used to filter out satisfied events of PSOA from raw event messages. The triggering condition manager checks the “AND” combination of the three events for PSOA\_0001. The service invocation manager invokes the “booking\_ticket\_service” for PSOA\_0001 if the combination of the conditions is satisfied.

\subsection{2.2 Installation Manager}

The PSOA installation manager on the upper left of Figure 2 consists of two key components, i.e., an installation interface and a code generator. The former enables users to define each stage of PSOA as nodes in a graph and the latter transforms users’ PSOA specification into multiple executable Java code corresponding to each part of PSOA.

Here is how the installation manager works. Firstly, a user uses the Web based GUI of the installation interface component to define a PSOA through XML. Secondly, the code generator extracts the users’ input from the XML-based PSOA specification using XSLT (the Extensible Stylesheet Language Transformations) [3] technology as shown in Figure 3. Thirdly, the code generator weaves the extracted users’ input with the pre-defined Java code templates to generate executable Java code. The generated Java code corresponds to the key functions, e.g., event monitoring, triggering condition evaluation, and service invocation. Finally the Java code will be compiled and deployed into the right place of the system by an installation script, which is also generated by the code generator.

By using the up-to-date code generation technique [15], we can streamline the whole process which has better efficiency and correctness than manual based methods.

\subsection{2.3 Event Filter Manager}

Event filter manager on the bottom left of Figure 2 filters out events that are related to the PSOA from raw event messages sent from various event monitoring wrappers. Event monitoring wrappers in the bottom of Figure 2 act as sentinels to collect, analyze, and transport raw event messages from event sources to the system. Events can be the content update from arbitrary web pages, instructions from enterprise management process, or the responses of web services invocation, depending upon the requirement and viewpoint of individual users. Event monitoring wrappers take the semantics of the corresponding event message information and transform the event message into predefined XML format (e.g., IBM Common Base Event format) messages, which in turn will be forwarded to event filter manager for filtering.

Event filter manager is extensible to other event processing and filtering systems. Existing event processing and filtering systems for some specific applications can be imported into our PSOA system directly. For example, RSS Reader and Atom reader [22] which are two typical event monitoring wrappers that monitor the update of web pages can be imported into our PSOA system directly. A Publish/Subscribe system can also be integrated into the PSOA system as a message-based event monitoring system.

All the events and the status of the events are stored by an active database. A database trigger is applied to each event as a piece of procedure code that is automatically executed in active databases in response to UDI operations (insert, update, delete) to a specific table. A database trigger for a PSOA works in the following steps. Firstly, the raw messages of each type of event are stored in a dedicated table in the database. Secondly, database triggers are used to filter out events by defining a predicate for one type of event. Finally, when there are UDI operations (insert, update, delete), the predicate is checked and the status of the events are stored in a fact table. The following example shows how a database trigger for checking the flight event of PSOA\_0001 works.

Firstly, we assume that PSOA\_FLIGHT is a table that stores flight update events as shown in Table 1. All raw flight event message received from the corresponding event monitoring wrapper will be inserted into this table.

\begin{table}[th]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Airline} & \textbf{Depart} & \textbf{Dest} & \textbf{Depart date} & \textbf{price} \\
\hline
US Airways & Atlanta & Baltimore & 2010-06-16 & 320 \\
\hline
Delta & Atlanta & New York & 2010-06-16 & 320 \\
\hline
Continental & Atlanta & Boston & 2010-06-16 & 310 \\
\hline
\end{tabular}
\caption{A PSOA\_Flight table}
\end{table}

Secondly, following PSOA\_0001, the flight condition is true if and only if there is a Delta flight from Atlanta to New York with fixed departure date (2010-06-16) which
costs less than $200. TRIGGER PSOA_0001_Flight is a database trigger that is registered on PSOA_FLIGHT. When a flight record is inserted, updated or deleted within PSOA_FLIGHT, all the database triggers registered in this table such as TRIGGER PSOA_0001_Flight will be checked to see whether the change of PSOA_FLIGHT satisfies the condition of the database triggers.

CREATE TRIGGER PSOA_0001_Flight
   After insert on PSOA_FLIGHT for each row
BEGIN IF
   (departure = 'Atlanta' and destination = 'New York' and departureDate = TO_DATE ('2010-06-16', 'YYYY-MM-DD') and fare \leq 200)
THEN
   UPDATE triggerResult SET checkbit = 1
   WHERE RuleName = 'PSOA_0001' and type = 'flight';
END IF;
END PSOA_0001_Flight;

Finally, the evaluation result will be stored in a separate fact table named TRIGGER_RESULT as shown in Table 2. As shown in TRIGGER_RESULT, we use a checkbit attribute to describe the satisfaction of an event. For example, the checkbit attribute of PSOA_0001 is 1 in the second tuple of the table, which indicates the “Flight” event for PSOA_0001 is satisfied. Note that the fact table needs to be truncated from time to time in order to keep an appropriate table size for the efficiency of trigger evaluation for the triggering condition manager. On one hand, if the fact table is too large, there is a significant overhead to process the table. However, on the other hand, if the fact table is too small, it may miss the time window when the combination of the events is already satisfied. The exact frequency of truncating a fact table is out of the scope of this paper and is left for future exploration.

<table>
<thead>
<tr>
<th>PSOA_ID</th>
<th>Condition type</th>
<th>Checkbit</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSOA_0001</td>
<td>Weather</td>
<td>0</td>
<td>2010-01-13 13:51:36</td>
</tr>
<tr>
<td>PSOA_0001</td>
<td>Flight</td>
<td>1</td>
<td>2010-01-15 13:24:36</td>
</tr>
<tr>
<td>PSOA_0002</td>
<td>Weather</td>
<td>1</td>
<td>2010-01-15 13:34:37</td>
</tr>
<tr>
<td>PSOA_0002</td>
<td>Flight</td>
<td>0</td>
<td>2010-01-15 13:24:36</td>
</tr>
<tr>
<td>PSOA_0003</td>
<td>Weather</td>
<td>1</td>
<td>2010-01-15 13:25:36</td>
</tr>
</tbody>
</table>

2.4 Triggering Condition Manager

Triggering condition manager evaluates the triggering conditions for all registered PSOA’s. For each PSOA, a triggering condition becomes true if and only if the predicate or the combination of the conditions is true. In our PSOA_0001 example, the triggering condition is $e_{\text{weather}} \land e_{\text{flight}} \land e_{\text{hotel}}$, which means the triggering condition becomes true only when all the weather, flight, and hotel events become true. The triggering condition manager fetches the checkbit of the rules related to $e_{\text{weather}} \land e_{\text{flight}} \land e_{\text{hotel}}$. The triggering condition is satisfied only when all the checkbits of the rules are 1.

It is the most important part of the whole PSOA system since multiple related events need to be evaluated concurrently. A triggering condition becomes true only when the related events satisfy a predicate or a combination of the conditions. Triggering condition management needs high scalability especially when a large amount of PSOA’s are registered in the system.

2.5 Service Invocation Manager

Service invocation manager on the right side of Figure 2 manages service invocation. It works according to the following steps. Firstly, Service invocation manager stores the metadata of candidate Web services(e.g. WSDL). Second, the user specifies the service invocation related information (e.g. credit card information to reserve an airline ticket) through PSOA Installation interface. Code generator extracts the information and use a service invocation template to generate service invokers. Finally, when the triggering condition is satisfied, the Service Invocation Manager picks up the right service invokers to execute. The response of the web services are returned to the user.

We also provide interfaces for service invocation manager to integrate with other web service invocation methods. For example, our service invocation manager can be integrated with IBM WSOM (Web Service Outsourcing Manager [23]), which is a framework that enables dynamic composition of Web service flow based on customer requirements. WSOM allows effective searching for appropriate Web services and integration of them into one composite Web service for performing a specific task. For another example, our Service Invocation Manager can also integrate with Message-based dynamic Web services invocation [11]. Message based service invocation technique requires that service invokers bind to arbitrary Web services at runtime. Service invokers and service providers are totally decoupled. In that case, users in our PSOA system just need to indicate the type of desired Web service (e.g. airline ticket reservation service) and service invocation manager will automatically search a web service that best matches users’ requirement at runtime.

3 PROACTIVE EVENT SERVICE VALIDATION

We use a formal method based on Petri nets to model and validate the correctness of a PSOA system.

3.1 Modeling PSOA with Petri nets

We summarize three important notations in PSOA.

1. Events. The events that cause a PSOA to fire can come from different resources: (a) the change of a database state caused by INSERT, DELETE, UPDATE operations; (b) the events that cause clock signals (e.g., check the balance of all bank accounts at 5:00pm every day); (c) any user- or application- generated signals (e.g. a successful order confirmation from Amazon.com). Furthermore, the...
triggering conditions to be monitored may be complex, and may be defined not only on single basic event such as single data value or individual database state, but also on sets of basic events with internal relationship between them (the expenditure of the travel plan should be smaller than the income of this month).

(2) Conditions. Triggering condition in PSOA can be divided into two types: time-based triggering condition and content-based triggering condition. Three types of temporal events are supported for time-based triggering condition: (a) absolute time points defined by the system clock (e.g., 5:00:00 pm, August 30, 2009); (b) regular time interval (e.g., execute every day at 5:00:00 pm); (c) relative temporal event (e.g., 30 seconds after an event occurred). A content-based triggering condition is defined upon content-based events, which can be expressed in terms of a database query, a built-in situation assessment function, or a user-defined method. Examples include: a simple condition on the database state (e.g. invoke whenever a deposit of $10,000 is made), an aggregate function on the database state (e.g. invoke whenever a deposit of $10,000 is made), an aggregate function on the database state (e.g. invoke whenever the price of an IBM X200 laptop drops below $800 at Amazon.com). Condition can also be composite ones which consist of multiple sub-conditions.

(3) Actions. Action specifies a set of web services to be invoked. The satisfaction of the corresponding condition can trigger the execution of a single service or multiple services with causal relationship between them. A service may perform simple functions, or trigger an entire business process. The response of invoked services can be delivered to end users or monitored by other PSOA’s as a content-based event. The response of invoked services can be delivered to end users or monitored by other PSOA’s as a content-based event. The latter case is more complex since the output of a PSOA can also be the input of another PSOA in a pipeline.

Three events involved in the PSOA_0001 can be represented as \( E_{set} = \{e_{weather}, e_{flight}, e_{hotel}\} \). The triggering condition in this example is the satisfaction of three subconditions. The invoking service is the ticket reserving service provided by a travel agency website. The stop condition specifies that the PSOA request should be removed when the system successfully invokes an airline ticket reservation service, or the PSOA request expires if the triggering condition is not satisfied in one month.

Definition 1: A Petri net \([24], [21], [7], [13]\) is a 3-tuple, \( N = (P, T, F) \) where:

\[
P = \{p_1, p_2, ..., p_m\}, m > 0, \text{ is a finite set of places pictured by circles;}
\]

\[
T = \{t_1, t_2, ..., t_n\}, n > 0, \text{ is a finite set of transitions pictured by bars;}
\]

\[
F \subseteq (P \times T) \cap (T \times P), \text{ is the incidence relation. Based on } F, \text{ we can derive the input and output functions which are } m \times n \text{ matrices.}
\]

Postset of \( t \) is the set of output places of \( t \) while preset of \( t \) is the set of input places of \( t \). \( M : P \to Z \), is a marking where \( M(p) \) represents the number of tokens in place \( p \) and \( Z \) is the set of non-negative integers. An initial marking is denoted by \( M_0 \). Tokens are pictured by dots. \((N, M)\) is called a net system or marked net. A transition \( t \in T \) is enabled under \( M \) if \( M(p) > 0 \) holds for every \( p \) in preset of \( t \). If a transition \( t \) is enabled, it may fire, resulting in a new marking \( M' \) with \( M'(p) = M(p) - 1 \) if \( p \) is in the preset of \( t \) but not in the postset of \( t \); \( M'(p) = M(p) + 1 \) if \( p \) is in the postset of \( t \) but not in the preset of \( t \); and otherwise \( M'(p) = M(p) \).

\( M' \) is reachable from \( M \) iff there exists a firing sequence such that by firing the transitions in the sequence, \( M \) can be changed to \( M' \). A Petri net \((N, M)\) is bounded iff for each place \( p \) there is a natural number \( n \) such that for every reachable state the number of tokens in \( p \) is less than \( n \). The net is safe iff for each place the maximum number of tokens does not exceed 1. A Petri net is strongly connected iff for every pair of nodes (i.e. places and transitions) \( x \) and \( y \), there is a path leading from \( x \) to \( y \).

Definition 2: A Petri net \( N = (P, T, F) \) is a WF-net (Workflow net) if and only if:

(1) It has two special places: \( \alpha \) and \( \beta \). Place \( \alpha \) is a source place: \( \alpha = \emptyset \). Place \( \beta \) is a sink place: \( \beta = \emptyset \).

(2) If we add a transition \( t' \) to \( N \) which connects place \( \beta \) with \( \alpha \) (i.e. \( t' = \alpha \) and \( t' = \beta \)), then the resulting Petri net is strongly connected.

By modeling the events and actions with places and modeling the condition with transitions, we can model the ECA chain with Petri nets. The main advantage of modeling the ECA chain with Petri nets is to be composed with the current work of web service models. As there are a plethora of research works which converts web service into Petri nets, the ECA chain models can be combined seamlessly with the web service models. In the PSSO_0001 example, the event of weather update information is the output of weather information service as well as the input of the ECA rule. The output of the ECA rule, i.e., the triggering of a booking flight service is exactly the input of the next web service. Then we can chain web services with ECA rules as shown in Figure 7.

Definition 3: An ECA chain is defined as

(a) a simple chain \( t \) where \( \alpha = \beta = \alpha \)

(b) an “AND” chain \( t \) where \( \alpha = \beta = \beta_1, \beta_2, ..., \beta_n \) and \( \alpha = \beta \)

(c) an “OR” chain \( t_1, t_2, ..., t_n \), where \( \alpha = \beta_i \) and \( \alpha = \beta \) for \( i = 1, 2, ..., n \)

We show how we model the ECA chain with Petri nets in Figure 4. We model a simple event-condition-action as shown in (a). When the event happens, there will be a token in the corresponding event place as shown in (b). Then the transition is enabled. When condition is met, the transition will fire, which will trigger the action. The final result is that there will be a token in the corresponding action place as shown in (c). We also show the simple, “AND” and “OR” ECA chains as shown in Figure 5 (a-c). For example, Figure 5 (b) models the “AND” condition. According to the enabling and firing rule of Petri nets, if and only if both of the event arrives, the transition will be enabled. Then the condition is checked and action is taken.

Definition 4: A PSOA is a 2-tuple \( N = (WF-nets, ECA) \).
chains) which denotes a composition of WF-nets and ECA chains:

1) WF-nets: WF-net \(1\), WF-net \(2\), ..., WF-net \(n\).
2) ECA chains: \(t_1\), \(t_2\), ..., \(t_m\).
3) The ECA chains connect the \(\beta\) of WF-nets with \(\alpha\) of other WF-nets.

According to our modeling steps, we can model the ECA chain of PSOA\_0001 with Petri nets as shown in Figure 6. We can chain web services of PSOA\_0001 as shown in Figure 7. Before we give the sound property for a workflow net and a PSOA, we give some definitions. We define state \(\alpha\) as \(M(\alpha) = 1\) and \(M(p) = 0\), \(\forall p \in P\{\alpha\}\). We define state \(\beta\) as \(M(\beta) = 1\) and \(M(p) = 0\), \(\forall p \in P\{\beta\}\). The initial state for a PSOA is defined as a marking where there is a token in the \(\alpha\) of all the event source WF-nets and there is no token in the other places. The final state for a PSOA is defined as a marking where there is a token in the \(\beta\) of all the expected-to-be-invoked WF-nets.

**Definition 5:** A workflow net for a web service is sound if and only if:

1) For every state \(M\) reachable from state \(\alpha\), there exists a firing sequence leading from state \(M\) to state \(\beta\).
2) State \(\beta\) is the only state reachable from state \(\alpha\) with at least one token in place.

**Definition 6:** A PSOA is sound if and only if there exists an initial marking leading from the initial state to the final state.

This definition implies that the correct web service is finally invoked (the final state) with the events from the available raw event sources (the initial state). This is exactly the objective of PSOA.

**Theorem:** A PSOA \(N=(WF-nets, ECA\ chains)\) is sound if all the WF-nets are sound and independent.

**Proof:** Consider an ECA chain in a PSOA. According to the definition of ECA chain, the input places of every transition \(t\) of this ECA chain, i.e., \(\ast t\) is a set of places of \(\beta\) from WF-nets. According to the definition of the soundness of WF-net [16], the place of \(\beta\) for this WF-net could be marked. If all the WF-nets are independent, there always exists a situation when all the \(\ast t\) are marked. Thus, every transition \(t\) of an ECA chain has a chance to be enabled. The firing of \(t\) will make the places of \(\ast t\) marked, which are a set of places of \(\alpha\) from WF-nets. By simple deduction, the final state of PSOA could be reached. Thus, the theorem holds.

According to the theorem, the soundness of the WF-nets is a sufficient condition for the soundness of a PSOA. Fortunately, the soundness property of a WF-net can be verified through many existing Petri nets tools and algorithms [16].

## 4 Performance Evaluation and Improvement

### 4.1 Experimental Setup

We deploy a prototype of our PSOA system into three servers for performance evaluation. The first server installs the main components of the PSOA system, including PSOA installation manager, PSOA controller, triggering condition evaluator, and service invocation manager. The second one installs the event filter manager with three event monitoring wrappers (the RSS reader in this case) that monitor weather, flight, and hotel update event. The third one installs Oracle10.2.2: an Database server supports multiple database triggers for each single table. The first and second server uses an Intel Core 2GHz Duo 2.66 GHz processor with 2 GB of memory and the third one uses a Pentium 1.7G with 2GB memory.

We implement a travel agency web service and deploy it in Tomcat 5.5 with Axis 2.0 on a separate machine.
to simulate the back end web services. In order to get real event streams, we utilize the RSS reader to monitor the updates from http://weather.yahoo.com/ to get the real weather information and http://www.orbitz.com/ to get the real flight and hotel information.

4.2 Performance evaluation

The performance of the system depends on certain chosen criteria, such as service latency, throughput, scalability, consistency, security, etc. In this paper, we focus on service latency, or response time and throughput. We define the service latency as the time between event detection and the service invocation. This is the end-to-end time that is spent in our PSOA. We do not count in the time to process the event stream as well as the execution time of the invoked Web services. The main reason is that such time strongly depends on many uncontrolled factors outside PSOA: the event source, the location of service providers, network traffic, and the complexity of the invoked Web services themselves. We also define throughput as the maximum number of requests per second that our system is able to handle.

Different registered events are related to different number of PSOA’s. For example, some hot event, e.g., weather prediction, flight information, and stock price update, are related to hundreds of thousands of PSOA’s. However some other events are only interested by a few PSOA’s. We focus on the average response time of the PSOA’s that are related to hot events since the hot events are more challenging for the system. In the following experiments, we take the weather event from weather.Yahoo.com as our testing event source. We scale up the number of PSOA’s when we use the grouping technique with group size of 5, 20, and 100 respectively. In the experiments we implement the triggering condition grouping technique using an index structure as shown in Figure 9. Every type of event follows a set of event instances with different triggering conditions. Triggering conditions in a group can share the same database trigger for their common event. In this way, triggering conditions in a group only needs to register one database trigger for their common event. As a result, \( T_{p-ses_1} \) and \( T_{p-ses_2} \) can be grouped together based on their common event \( e_1 \). According to our grouping method, \( T_{p-ses_1} \) and \( T_{p-ses_2} \) can share the same database trigger for \( e_1 \). In this way, triggering conditions in a group only needs to register one database trigger for their common event in the underlying database. This grouping strategy can significantly reduce the number of database triggers especially when a large number of triggering conditions are related to the same hot events, e.g., the weather event.

We implement the triggering condition grouping technique using an index structure as shown in Figure 9. Every type of event follows a set of event instances with different triggering conditions. Triggering conditions sharing any common event are grouped together under the same list of the common event. For example, PSOA0001 and PSOA0003 are in the same group by sharing the same weather event. For example, when there are 1000 registered PSOA requests, the response time is 2 seconds when 300 PSOA triggering conditions are satisfied, compared with 5 seconds when 900 PSOA triggering conditions are satisfied. The main reason is that, when the triggered PSOA’s increases, the load to service invocation manager also increases. This further increases the response time.
assume 300 PSOA’s are triggered among the total number of registered PSOA’s for weather event. Compared with previous performance, we can see an improvement as the average response time drops. For example, when the number of registered PSOA’s is 3,000, the average response time with a group size of 5 is 4 seconds. It is 2 seconds for a group size of 20 and less than 1 second for a group size of 100. However, it is 8 seconds without grouping.

Figure 11 shows the throughput of the system for different grouping size. The improvement is also clear. For example, when there are 3,000 PSOA’s that are registered for weather event, the throughput is 450 per second without grouping. However, the throughput is 750 when group size is 5.

5 RELATED WORK

Publish/subscribe: the main purpose of traditional publish/subscribe systems is to fully decouple communicating entities in time, space, and synchronization [6], [20], [10]. A publish/subscribe system routes messages from publishers to subscribers who are interested in these messages according to some specified mechanisms, such as queue-based, topic-based, content-based, and type-based [6]. One significant difference between our PSOA system and Publish/Subscribe systems lies in the different application domain. Publish/subscribe systems route messages from publishers to subscribers by specifying specific filtering rules. Users in our PSOA system do not focus on retrieving information from information providers, but focus on taking actions in time based on the analysis of events interested by users (e.g. invoke an airline ticket reservation service when the ticket price falls below $50).

Active databases: Most active database systems [17] provide facilities that allow users to specify, usually in the form of database triggers, actions to be performed following the changes of database states. The purpose of database triggers is to achieve automatic database management [17]. Despite their conceptual generality, database triggers have been so far supported in a fairly restrictive form in practical systems. For example, the limitations of the monitoring events (built-in SQL operations such as update, insert, delete) on a single table and actions to be taken (update, insert, delete, or the invocation of a stored procedure).

PSOA’s applies in a more complex environment, since they can be defined on multiple heterogeneous event sources, and the action scope of PSOA is the SOA environment, not limited to the database itself.

Web Service eventing and notification: Web Service Eventing (WS-Eventing) and Web Service Notification (WS-Notification) [8], [1] are two industrial standards to define a set of specifications that standardize the way web services interact with each other using notifications or events. They both follow the publish/subscribe paradigm that a Web service subscribes to or accepts subscriptions for event notification messages from another Web service through a broker service (mediation system in Publish/Subscribe). The main purpose of WS-Eventing and WS-Notification is to enable the asynchronous communication among different entities in the SOA environment. They mainly focus on events or messages transporting and routing between different Web services while neglecting heterogeneous event monitoring and triggering condition evaluation, which is necessary to build a responsive and intelligent SOA environment.

Event-driven SOA in industry: Major players in industry, including IBM, BEA, and Oracle has shown interest in event-driven SOA. These companies are now beginning to bring event-driven computing into mainstream IT environments by embedding events into existing SOA deployments. However, their event-driven computing is just applied to limited applications, and not available on a broad scale [4]. Our PSOA system addresses this problem by providing a flexible and pluggable framework for the easy combination of event monitoring, trigger evaluation, and services invocation. In addition, a PSOA requires explicit specification of stop condition. Introducing stop condition as a necessary component of PSOA is to avoid serious side-effect such as over invocation (invoke a proactive SOA event service when it is already expired) or endless cascading invocation (invocation results of a PSOA can be update events which may in turn trigger the same PSOA again, directly or indirectly).

6 CONCLUSION

Following SOA 2.0, we implement a proactive service-oriented architecture (PSOA) system. The system consists
of three main components. The first component, an event stream filtering system, filters out events that are related to the PSOA. The second component, triggering condition processing, handles the evaluation of triggering conditions. The third component, service invocation system, invokes the web services.

Beyond the design and implementation of proactive service-oriented architecture (PSOA) system, we focus on the validation and performance improvement. We propose a formal method based on Petri-nets to model and validate the soundness of the proactive service-oriented architecture (PSOA) system. We propose a group-based approach to deal with the performance of the system with respect to the response time and the throughput under large-scale situation. Experimental results show that the response time decreases and the throughput increases after the group-based approach is adopted. As one of our future work, we plan to integrate PSOA system as an excellent node in the context of social networking when the services interact with others.

References

Calton Pu received his PhD from University of Washington in 1986 and served on the faculty of Columbia University and Oregon Graduate Institute. Currently, he is holding the position of Professor and John P. Imlay, Jr. Chair in Software at the College of Computing, Georgia Institute of Technology. He is leading the Infosphere project, building software tools to support information flow-driven applications such as digital libraries and electronic commerce. Infosphere builds on his previous and ongoing research interests. First, he has been working on next-generation operating system kernels to achieve high performance, adaptiveness, security, and modularity, using program specialization, software feedback, and domain-specific languages. This area has included projects such as Synthetix, Immunix, Microlanguages, and Microfeedback, applied to distributed multimedia and system survivability. Second, he has been working on new data and transaction management by extending database technology. This area has included projects such as Epsilon Serializability, Reflective Transaction Framework, and Continual Queries over the Internet. His collaborations include applications of these techniques in scientific research on macromolecular structure data, weather data, and environmental data, as well as in industrial settings. He has published more than 60 journal papers and book chapters, 150 conference and refereed workshop papers, and served on more than 90 program committees, including the co-PC chairs of SRDS’95, ICDE’99, CoopIS’02, SRDS’03, DOA’07, and co-general chair of ICDE’97, CIKM’01, ICDE’06, DEPSA’07, CEAS’07.