

A XML-Based Composition Event Approach as an Integration and Cooperation Middleware

Gang Xu, JianGang Ma, and Tao Huang

Technology Center of Software Engineering, Institute of Software,
Chinese Academy of Sciences, Beijing 100080, P.R. China
{xugang, mjg, tao}@otcaix.iscas.ac.cn

Abstract. Content-based Pub/Sub systems are gaining increasing popularity as an integration and cooperation middleware supporting large-scale distributed applications, especially for Grid. However, existing content-based Pub/Sub systems mainly focus on the “one-to-one” event matching, not on the composite event matching. The composite event matching enables application components to express more interest in the occurrence of event composite patterns. In this paper, we present a XML-based composite event model that consists of the temporal logical model and the event composite pattern. Based on the composite event model, a subscription language (EXML-QL) is introduced by extending XML-QL to support the XML-based composite event computing. Finally, in terms of the peculiarities of the subscription language, a composite event matching process is presented and analyzed.

1 Introduction

As a loose coupling and scalable integration middleware, content-based Pub/Sub systems are gaining increasing popularity in the area of large-scale distributed applications, such as Grid [1]. However, existing content-based Pub/Sub systems only support the single event matching and lose the ability to express the more interests of event composite patterns. For example, in Grid information integration, information formatted as XML must be delivered timely, which raises the potential need that the users only receive relevant information under certain event conditions, not the whole huge XML data.

To solve this problem, we propose a XML-based composite event model and its subscription language. The composite event model describes the temporal relationship of composite events and the event forwarding patterns. At the same time, its subscription language supports the composite operations on XML events.

2 Composite Event Model

The composite event model of content-based Pub/Sub systems consists of the event temporal logic models and the event composite patterns. The event temporal logic models describe the time sequence relationship of events in distributed environment; the event composite patterns describe the event forwarding patterns.

2.1 Temporal Logic Model

In distributed environments, we use a two-part interval timestamp to illustrate events, rather than a single conventional timestamp due to network delays or radio transmission lag. In terms of the characteristics of interval timestamps, the temporal logic relation definition is defined as follows:

Definition 1: $t = [t^s, t^f]$ denotes an interval timestamp of events, where t^s denotes the time at which publishers release events; t^f denotes the time at which Pub/Sub systems receive events. Temporal logic relation is $T-Logic \in \{<, \prec, \supset, \cup\}$, where “ $<$ ” denotes the total order relation, “ \prec ” denotes the partial order relation. Temporal logic relations meet the following formal relations: $t_i < t_q \Leftrightarrow t_i^f < t_q^s$;
 $t_i \prec t_q \Leftrightarrow (t_i^s < t_q^s < t_i^f) \vee (t_i^f < t_q^f)$; $t_i \supset t_q \Leftrightarrow (t_i^s < t_q^s < t_i^f) \vee (t_q^f < t_i^f)$;
 $t_i \cup t_q \Leftrightarrow [\min(t_i^s, t_q^s), \max(t_i^f, t_q^f)]$.

2.2 Event Composite Patterns

Event composite patterns illustrate different event matching and forwarding approaches. According to the relation of the matched content and the forwarded content and the relation of the in-degree and the out-degree of Pub/Sub systems, we define the event composite patterns as follows:

Definition 2: Event composite operator: $Composite-Operator \in \{\xrightarrow{=}, \xrightarrow{=}, \xrightarrow{\supset}, \xrightarrow{\subset}\}$, where “ $\xrightarrow{=}$ ” denotes “forwarding pattern” that is correspondent to the “one-to-one” event matching, namely it resolves a single input event matching and forwards this single event. “ $\xrightarrow{=}$ ” denotes “triggering pattern” that is correspondent to one of the “many-to-one” event matching. For the “triggering pattern”, Boolean evaluating the temporal logic relations of several input events is the pre-condition. The events that be forwarded may be arbitrary input events. “ $\xrightarrow{\supset}$ ” denotes “splitting pattern” that is correspondent to the “one-to-many” event matching pattern. Other than matching the single input event, the “splitting pattern” is required to have the capability of select the particular part of the event’s content, forwarding the selected contents to one or several subscribers. “ $\xrightarrow{\subset}$ ” denotes “merging pattern” that is correspondent to the other of the “many-to-one” event matching pattern. The “merging pattern” needs to merge several events distributed over the temporal sequence into a single event, and further forward this merged event.

3 Composite Event Subscription Language

In Pub/Sub systems, information providers publish information in the form of events, information consumers subscribe to events by subscription languages, and Pub/Sub systems deliver the published events to all interested subscribers. Events and subscription languages are an interactive entity. In other words, what form of events decides the form of subscription languages; what form of subscription language also decides the form of events.

Without loss of generality, we define an XML-event as below:

Definition 3: an XML event is a tri-tuple: XML-event=(element, attribute, structure), where elements and attributes have name, type and value, attributes belong to certain elements, structures restrict the structural relations of elements, such as parent, sibling, ancestor.

We further define the temporal operators in the events as the following:

Definition 4: Temporal operator: T-Operator $\in \{“:”, “;”, “||”, “\perp”, “\nabla”\}$, where “:” denotes the concatenation event that follows the partial order relation and overlaps in time, such as $e_1:e_2$; “;” denotes the sequence event that follows the complete order relation and doesn’t overlap in time, such as $e_1;e_2$; “||” denotes the parallelization event that matches two events in parallel, and succeeds only if both are matched. Any order is allowed and events overlap in time, such as $e_1||e_2$; “ \perp ” denotes the timing event that matches a event within a give interval, such as $(e_1 \perp e_2)e_3$; “ ∇ ” denotes the alternation event that is satisfied if either e_1 or e_2 is matched, such as $e_1 \nabla e_2$.

In the environment of the unrestricted context, Pub/Sub systems may match several input events and not all input events may be meaningful to the event forwarding. For example, for the sequence event and the triggering operator $(e_1;e_2 \xrightarrow{=} e_3)$, the several events of e_3 may be matched after e_1 happened and before e_2 happens. So, we need to further define parameter contexts to designate which event to be used.

Definition 5: Parameter-Context $\in \{\text{Recent, Exclusive, Globe}\}$, where “Recent” represents only the most recent input event is used; “Exclusive” represents only the events that happen after the terminating event is matched are used, “Globe” represents the events that happen after the initiating event is matched are used.

It is easy to prove that the synthesis of temporal logic evaluations and parameter contexts is the fairness assertion of the composite event temporal logic.

3.1 Composite Event Subscription Language EXML-QL

In the section, we extend XML-QL to be a composite event subscription language EXML-QL (Event XML-QL). XML-QL itself is a declarative “relational complete” query language and is simple enough that it can be optimized [2]. XML-QL is suitable for extracting data from existing XML documents and constructing new XML documents. In the aspect of constructing new XML documents, XML-QL utilizes nested queries to support the structure hierarchy of XML data and

```

WHERE<$><author><firstname>$fn</>
  <lastname>$ln</>
</>
<title>$t</>
</>IN"www.a.b.c/bib.xml"
CONSTRUCT<OutputXML>
<person ID=PersonID($fn,$ln)>
  <firstname>$fn</>
  <lastname>$ln</>
  <publicationtitle>$t</>
</>
</>
    
```

Fig. 1. An XML-QL example

use Skolem functions to control how the result is grouped. A XML-QL example is given in Fig. 1.

In the example, variable names are preceded by \$ to distinguish them from string literals. The “WHERE” clause contains the content needed to be matched and extracted. The “CONSTRUCT” clause constructs a XML document whose root element is a <person> element. Among the “CONSTRUCT” clause, PersonID is a Skolem function that can produce a new tag for each new (\$fn, \$ln).

Though XML-QL has many advantages in constructing XML documents, it can’t satisfy the need of composite event subscription languages as follows:

- In Pub/Sub systems, detecting events is based on the content of events, so it is impossible to name events in advance.
- In XML-QL, if omitting “IN” clauses, the XML matching sections for different events can’t be distinguished.
- XML-QL doesn’t contain the time concept, so it doesn’t support the temporal logic evaluation.
- XML-QL doesn’t support the event composite patterns.

Based on the existing function of XML-QL, we further define EXML-QL by adding the time operators and the composite operators, using the “block-variable” substituting the “IN” clauses in XML-QL to support the composite event subscription. EXML-QL is defined as follows:

Definition 6: EXML-QL is a extended XML-QL language for the composite event subscription: $EXML-QL = \{ \#Block-variable \} \cup \{ T-Operator \} \cup \{ Composite-Operator \} \cup \{ XML-QL \}$, where # is the prefix of block variables, T-Operator is the time operators, Composite-Operator is the composite operators.

So, we change the grammar of XML-QL as the following.

- $EXML-QL ::= (Function \mid Query \mid CompositeCondition) \langle EOF \rangle$;
- $CompositeCondition ::= Block-variable \mid T-Operator \mid EP$;
- $Condition ::= Pattern \text{ BindingAs } * \text{ ‘\#’ } Block-variable \mid Predicate$.

Fig. 2 is a simple EXML-QL example. This example contains two block variables #P and #T representing the <person> element and the <orderpayer> element from the different events. In the end of the “WHERE” clause, we add the temporal operators and utilize the block variables referencing to events to build the temporal logic relations of discrete events. This temporal logic relation in the example is a sequence event, namely the event #P weekly follows the event #T. The “CONSTRUCT” clause uses the content of the event #P to construct a new output XML event.

```
WHERE<person>
  <name></ELEMENT_AS $n
  <personnumber>$pn</>
</>#P
<orderpay>
  <pay>finished</>
  <deliver>ok</>
  <pn>$pn</>
</>#T
#T,#P
CONSTRUCT<result><pn>$pn</>$n</>
```

Fig. 2. An example of EXML-QL

4 Composite Events Matching

4.1 Composite Events Matching

Based on EXML-QL, matching composite events has the following characteristics:

- Decomposing the “WHERE” clause into a set of block variables and implementing the XML event matching.
- Based on the temporal logic built from the block variables, the temporal logic relations are evaluated on the matched events.
- Transforming the XML events that satisfy the temporal logic relation into the naming XML document, furthermore transforming EXML-QL into XML-QL and using XML-QL interpreters to construct new output XML events.

In the process of matching XML events, we convert block variables to Finite State Machines (FSM). The events that drive the execution of the matching are generated by the XML parser. In the execution model, a subscription is considered to match a XML event when the final state of its FSM is reached. We construct a main element index table that indexes all the subscriptions. The main elements are the root elements in block variables. The matching process is based on the “Push” mechanism, namely XML event are used to match subscriptions. For each parsing event, we implemented three functions to deal with the element start tag, the element end tag and the element internal data. Firstly, the main element index table is checked to match the root element in block variables; secondly, other element nodes are performed three checks: the name check, the level check and the predicate check. The name check makes sure that the parsing element name equals the name of the node. The level check makes sure that the level of the element appeared in the XML event matches the level of element in EXML-QL. The predicate check makes sure that the values of the element and attribute satisfy the evaluation of the matching predicates.

For the “WHERE” clauses that may contain the temporal logic, we construct event graphs to evaluate the composite event temporal logic. An event graph is a rooted, directed acyclic graph whose each node has an out-degree edge and two in-degrees edges and is labeled by a time operator. The terminal output of the whole event graph is “true” or “false” representing whether the XML event is triggered or not, or whether a new XML event is constructed or not. The non-terminal outputs of event graph are the union calculation of the time of the two input events. Besides, each node is appended a parameter context to restrict the selection of input events.

In the process of constructing new XML events, currently we only utilize the existing XML-QL interpreter and do the following change:

- transforming EXML-QL into XML-QL;
- Naming the XML events that satisfy the content matching and the temporal logic evaluation and input these named XML events into XML-QL interpreter;
- XML-QL interpreter constructs new output XML events.

4.2 Performance Analysis

Fig.3 is the result of calculating the matching rate of the sequence event. In the matching rate experiment, we restrict the EXML-QL only contains one sequence temporal

operator and construct a new output event in the “CONSTRUCT” clause. The number of subscription is 500; the whole number of Block variable is 1000. We have the 500 input XML events that are divided into ten groups of XML events, where only four groups meet the subscriptions. Each group of XML event has the same 50 XML event. The 500 input XML events are inputted in an average time interval way. Fig. 3 shows the event matching rate depends on the event input sequence and the matching rate increased when the number of the accumulated events increased.

In calculating the matching time of the sequence event, we restrict the EXML-QL only contains one sequence temporal operator, does not contain the “CONSTRUCT” clause and uses the event composite pattern operator to point out the event that is need to be forwarded. The number of block variables in each subscription is two; the number of subscriptions is from 500 to 10,000. Fig. 4 shows the result of the experiment that denotes the matching time is mostly proportional to the number of input events.

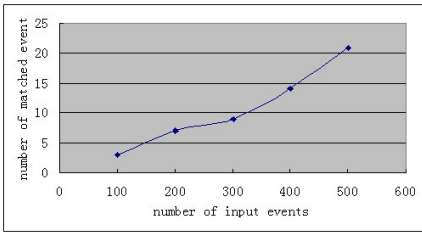


Fig. 3. Results of matching rates

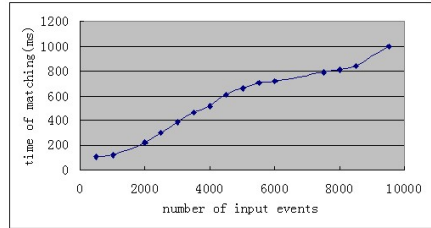


Fig. 4. Results of matching time

5 Related Works

Currently, XML-based Pub/Sub systems are mainly XFilter [3], YFilter [4], XTrie [5] and WebFilter [6]. They adopt XPath as subscription languages to performing efficient filtering of XML documents for large-scale information dissemination systems. XPath has more expressive capability for user interests than XML-QL, but XPath has not ability to construct new XML document. Compared to EXML-QL, XML-QL can't support composite event operations.

The researches of composite events mainly focus on active databases. Snoop was to design an expressive composite event specification language with powerful temporal support [7]. Peter discussed composite event detection in distributed environments [8]. Compared to this paper, it emphasizes particularly on the topological structure analysis of composite event detection in distributed environments and don't make reference to composite patterns such as splitting and merging.

6 Conclusions

We present the study of XML-based composite event matching approach, an important problem in next-generation, scalable content-based Pub/Sub systems. We introduce the temporal logical model to describe the temporal relationship of events in

distributed environments and the event composite pattern to describe the different forwarding and triggering patterns. Based on composite event model, a description language (EXML-QL) is presented by extending XML-QL. Finally, according to the peculiarities of the subscription language, the process of matching composite events is presented and further analyzed.

Acknowledgements. This work is partially supported by the Chinese National “973” Key Research Program (2002CB312002, 2002CB312005), the Chinese National “863” High-Tech Program (2001AA113010, 2001AA414020, 2001AA414330).

References

1. Zhuge H.: China's E-science knowledge grid environment. *IEEE Intelligent Systems* 1 (2004) 13-17
2. Deutsch A., Fernandez M., Florescu D.: XML-QL: A query language for XML. [Http://www.w3.org/TR/NOTE-xml-ql/](http://www.w3.org/TR/NOTE-xml-ql/), 1998
3. Altinel M., Franklin M. J.: Efficient filtering of XML document for selective dissemination of information. In: *Proceedings of 26th International Conference on Very Large Data Bases*, Cairo, Egypt (2000) 53-64
4. Yanlei Diao, Peter Fisher, Michael J. Franklin, Raymond T.: Yfilter: efficient and scalable filtering of XML documents. In: *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA (2002) 341-342
5. Chan C. Y., Felber P., Garofalakis M., Rastogi R.: Efficient filtering of XML documents with XPath expressions. *The VLDB Journal* 4 (2002) 354-379
6. Pereira J., Fabret F., Llirbat F., Jacobsen H. A., and Shasha D.: WebFilter: A high throughput XML-based Publish and Subscribe system. In: *Proceedings of 27th International Conference on Very Large Data Bases*, Roma, Italy (2003) 1101-1104
7. Chakravarthy S., Mishra D.: Snoop: An expressive event specification language for active databases. Technical Report UF-CIS-TR-93-007, Department of Computer and Information Science, University of Florida, Florida (1993)
8. Peter R. Pietzuch, Brian Shand, Jean Bacon: Composite event detection as a generic middleware extension. *IEEE Network* 1 (2004) 44-55