Handling Hierarchical Events In An Internet-Scale Event Service

Justin R. Erenkrantz
jerenkra@ics.uci.edu

25th March 2001

Abstract

During the course of developing a generic library for converting XML-based events to SIENA events, an issue was discovered concerning repetition of event attribute names. This paper will examine the inherent problem and present some possible solutions to the problem. This paper will first examine the relevant technologies, the implementation of the library, and then discuss potential solutions to the problem of mapping hierarchical events onto non-hierarchical namespaces.

Contents

1 Introduction

This research was initially conducted to develop an end-to-end demo of the SIENA system. Prior to commencing this research, no independent software existed which took advantage of the SIENA framework. For this particular application, it was decided to provide a tool which could render XML documents into a format suitable for distribution on a SIENA network.

During the course of developing such an application, an issue inherent in the design of the SIENA system had to be addressed. This fundamental issue concerns mapping a hierarchical event namespace onto a non-hierarchical event namespace. This paper will first examine the relevant SIENA and XML technologies. Then, the implementation of the SIENA-XML interface will be discussed. Finally, some possible alternatives and thoughts will be introduced for dealing with mapping inherently hierarchical events onto inherently non-hierarchical namespaces.

The implementation of the SIENA-XML interface may be found at the SIENA-XML Homepage `http://www.ucf.ics.uci.edu/~jerenk/siena-xml/`.

2 SIENA Events

SIENA (**S**calable **I**nternet **E**vent **N**otification **A**rchitecture) is an architecture designed for Internet-wide distribution of events [Car98]. As discussed in [CRW00], a tenuous balance exists between expressiveness and scalability in an Internet-scale event service. Since SIENA's primary design goal is scalability rather then expressiveness, SIENA is implemented with a flat event namespace. SIENA event names have no correlation with each other. Since SIENA does not have a hierarchical event namespace, hierarchical events (such as XML-based events) must therefore be translated into a flat namespace before transmittal on a SIENA network. This section will describe the layout and format of SIENA events and provide some examples of SIENA events.

2.1 Event format

SIENA events are a collection of attribute name-value pairs which serve to describe the underlying event. Multiple distinct attribute names can exist in the same SIENA event. An example SIENA event would look like:

this.is.an.example.of.a.siena.attribute.name	This is an example of a SIENA attribute value
ThisIsAnotherSienaAttributeName	42

Legitimate attribute names and values in a SIENA event may only contain alphanumeric characters, an underscore, a period, and a forward slash. Therefore, any events originating from other sources must have any illegal characters removed before transmittal on a SIENA-based network. Attribute names can not be repeated in a single SIENA event.

It is crucial to note that attribute names have no explicit relationship to each other. The relationship between two attribute names is only defined by the event source. For example, an event consisting of the following attributes:

stock.nyse.xyz.sharessold	100
stock.nyse.xyz.stockprice	50

The application sending or receiving the event may interpret a relationship between the two attributes, but there is no such correlation stored within the event format of SIENA. Furthermore, the . in the above attribute names serves only cosmetic purposes to help the human interacting with SIENA to easily parse the attribute names. The periods or slashes within the attribute name do not specifically serve any purpose when routing the messages.

The attribute values of a SIENA event can take on the following types: null, string, long integer, integer, double, and booleans. The current implementations of SIENA require that the published event and filters correspond to the same type. Therefore, a boolean type can not be compared to a long type. Caution must be taken when creating filters to ensure that they are of the same type as the event that will be published.

2.2 The publish-subscribe model

SIENA is implemented as a publish-subscribe event model with advertising. A client is made aware of an event via an advertisement event which details the structure of subsequent events and is broadcast to all connected clients. If a client wishes to listen to events of this advertised format, a subscription for this particular class of event can then be submitted via the SIENA network. When subscribing to an event, a set of filters can be included with the subscription. When an actual event occurs, the event is published and all subscribed client with matching filters receive the event.

2.3 Filters

The filters included with a subscription determine which events a client will receive. The scalability of SIENA is achieved by leveraging these filters in a collaborative manner. The filters for a subscription form the basis for the network topology of the overall SIENA system. By combining multiple subscriptions, the SIENA server will attempt to reduce the components to the smallest possible number of subscriptions while retaining the complete list of recipients and their respective filters.

A filter has the form of attribute name, constraint operator, and constraint. The attribute name listed in the filter must match exactly with the attribute name in the actual published event. Since wildcard filtering does not exist in SIENA, a filter that might apply to multiple attribute names must be explicitly specified for each attribute name. All component filters listed in a single filter are treated as a boolean AND - this means that all given filters must match in in order for the SIENA event to be passed on to the client.

The degree of complexity that can be expressed with these operators form the basis for the expressiveness and scalability of SIENA. The list of currently supported operators is listed below:

Siena Code	Description
EQ	Equal
LT	Less than
GT	Greater than
GE	Greater than or equal to
LE	Less than or equal to
PF	String prefix
SF	String suffix
XX	Always matches
NE	Not equal
SS	Substring

2.4 Examples

An example of a filter that a client could submit to a SIENA server would be:

this.is.an.example.of.a.siena.attribute.name	SS	“This”
ThisIsAnotherSienaAttributeName	LE	25

If the following event were published, the above filter would indicate a match and the event will be sent to the client:

this.is.an.example.of.a.siena.attribute.name	“This is an example of a success”
ThisIsAnotherSienaAttributeName	20

However, it would not match the following event because the attribute “this.is.an.example.of.a.siena.attribute.name” does not contain the substring “This” in the corresponding attribute value:

this.is.an.example.of.a.siena.attribute.name	“His story was good”
ThisIsAnotherSienaAttributeName	15

Furthermore, the following event would also not be detected because it does not contain the required “ThisIsAnotherSienaAttribute” attribute:

this.is.an.example.of.a.siena.attribute.name	“This story was good”

3 XML-based Events

As XML has matured, a number of prominent data sources based in XML have developed. XML (e**X**tensible **M**arkup **L**anguage) is an ideal candidate for certain types of data exchange. News sites such as CNN, magazines such as Salon and Wired, financial sites such as Motley Fool, and popular web sites such as Slashdot provide XML digests of their websites on a routine basis [XML]. By examining these XML documents, the current stories and information from these sites can be easily parsed. These documents form the basis of the event sources used by the Siena-XML project.

3.1 Constraints

One distinguishing feature of XML is that it does not place constraints upon the data that can be represented by an XML document, but rather provides a standard way of defining the structure by which a document can be defined. Only documents conforming to their corresponding DTD (**D**ocument **T**ype **D**efinition) can be considered valid XML documents. If a document does not conform to the standard representation, the parsing of the document is not guaranteed.

3.2 Inherent hierarchy and unknown multiplicity

Since XML is a derivative of SGML, it is hierarchical in nature. The power of XML resides in that any document must conform to the defined hierarchy, or it is invalid. An XML document’s DTD serves to define the relationship between elements in a document. However, the language of the DTD does not allow the event originator to explicitly specify the number of children present in a specific XML document (although it can often provide a lower-bound). An example of a section of a DTD is:

```
<!ELEMENT Z (A, B+, C?, (D|E))>
```

This particular DTD defines an element called Z which has five possible children - A, B, C, D, and E. Based upon the rules of the DTD, each element Z must have one A child, at least one B child, at most one C child, and must have an either a D or an E child (but not both). An example of a conforming XML document would be:

```
<Z attribute=“00”>
                    <A attribute=“11” />
                    <B attribute=“22” />
                    <B attribute=“33” />
                    <E attribute=“44” />
</Z>
```

Until encountering a particular XML document, it is not possible to know how many particular elements it may have. This presents the fundamental problem for mapping an XML document into a format suitable for SIENA.

3.3 Parsing strategies

When designing the XML components for this project, it was decided to attempt to make the XML parsing as generic as possible. Therefore, the XML parser does not assume anything about the document other than what was provided in the document in the form of DTDs. The goal of the SIENA-XML parsing is to parse any valid XML document into a corresponding SIENA event format. This design allows the knowledge of the data to be maintained at the event-source level, rather than placing application-layer logic in the parsing routines. As will be discussed later on, by making the parser more specific, it becomes possible to resolve some of the namespace problems with SIENA and XML. However, for the scope of the SIENA-XML project, the XML parsing will be generic.

4 SIENA-XML API

The purpose of this research project was to define a way to translate events from generic XML documents into SIENA-suitable events. For ease of implementation, Java was chosen as the language of choice. At the time the research project commenced, the Java API of SIENA was still in development. Therefore, this project also served as a mechanism to provide feedback to the original authors about the Java implementation of SIENA. The XML parsing was originally performed by the Java Project X Technology Release 2, but has since been updated to use the now-released reference implementation of JAXP (Java API for XML Parsing). The user interface was implemented using standard Swing components.

In developing this API, an attempt was made to isolate the components into one of three discrete categories:

1. SIENA specific components
2. XML specific components
3. User-interface specific components

Every attempt was made to ensure that the SIENA-specific components did not rely upon the XML-specific components to operate correctly, and vice versa. Each class should be able to stand-alone or operate via well-defined interfaces or data structures.

4.1 A Java-based package for handling XML-based events

In developing a Java-based XML event handler for SIENA, a series of Java framework classes were developed. This SIENA framework is based on the provided SIENA Java API, but simplifies the interaction between the developer and the underlying event service. The XML framework is based on the Sun JAXP 1.0 standard - any JAXP 1.0 compliant-parser can be used to parse the XML files. The core classes described herein are utilized by the Swing-based UI programs. The following sections provides a high-level description of the framework.

4.1.1 com.erenkrantz.siena.HashList

This data structure is implemented to appear identical to a traditional *java.util.Hashtable*, but it allows duplicate entries to be stored within the data structure. This implementation is based on the *java.util.Hashtable* and the *java.util.List* classes. This data structure is common between the SIENA-specific classes and the XML-specific classes.

4.1.2 com.erenkrantz.siena.SienaConnection

The *SienaConnection* class wraps the *HierarchicalDispatch* class provided by the SIENA API. This class contains all logic for connecting to and communicating with the SIENA server.

4.1.3 com.erenkrantz.siena.SienaSender

The *SienaSender* class extends the *SienaConnection* class and provides the following mechanisms:

1. Creating an *Event* object
2. Populating an *Event* object with the required attribute names and value
3. Sending an *Event*

4.1.4 com.erenkrantz.siena.SienaReceiver

The *SienaReceiver* class extends the *SienaConnection* class and provides the following mechanisms:

1. Setting a *Filter* for listening
2. Setting a callback routine for a *Filter*
3. Allowing subscriptions of a *Filter*
4. Allowing unsubscriptions of a *Filter*

4.1.5 com.erenkrantz.siena.xml.SienaXMLRenderer

The *SienaXMLRenderer* class will parse any XML document via a filename or URI into a *HashList*. It utilizes the JAXP 1.0-compliant parser to turn the XML document into a *HashList*. This *HashList* can then be distributed across the SIENA network via the *SienaSender* class.

4.1.6 com.erenkrantz.siena.xml.SienaXMLRenderFrame

The *SienaXMLRenderFrame* is the main user interface for interacting with rendering and transmittal of an event. It is responsible for loading an XML document via *SienaXMLRenderer* and rendering the document in the correct user-interface components. This class should then be able to transmit a rendered form of the original XML document via a *HashList* across the SIENA network by utilizing the *SienaSender* class.

4.1.7 com.erenkrantz.siena.xml.SienaXMLReceiveFrame

The *SienaXMLReceiveFrame* is the main user interface for interacting with the filtering and receival of an event. It is responsible for composing a proper *Filter* instance and marshaling that information to the appropriate *SienaReceiver* instance. Upon receiving an event, it should update the corresponding user-interface components.

5 Parsing of XML documents into events

The following section describes the process by which the SIENA-XML system translates an XML document into a format suitable for transmission across a SIENA network. The mechanism described herein is completely generic and does not account for any specific characteristics of an XML document.

5.1 Parsing the XML document

Once an XML document has been validated as being correct, it must be converted into a format suitable for the event system.

5.1.1 Simple XML Example

A simple XML document might look like the following:

```
<Document>
        <XElement xattribute="1" />
</Document>
```

Under the currently implemented mechanism, it would be translated into the following SIENA event attributes and values:

Document.XElement.xattribute	1

5.1.2 XML Example with Hierarchical Events

An example XML document might look like the following:

```
<Document>
        <XElement xattribute="1" />
        <YElement>
                <ZElement zattribute="2" />
        </YElement>
</Document>
```

Under the currently implemented mechanism, SIENA-XML would translate the document into the following SIENA event attributes and values:

Document.XElement.xattribute	1
Document.YElement.ZElement.zattribute	2

5.1.3 XML Example with Duplicate Elements

An example XML document might look like the following:

```
<Document>
        <XElement xattribute="1" />
        <XElement xattribute="2" />
</Document>
```

Under the currently implemented mechanism, it would be translated into the following SIENA event attributes and values:

Document.XElement.xattribute	1
Document.XElement.xattribute	2

5.2 Problem with Parsing

The example in Section 5.1.3 displays the fundamental problem with mapping the hierarchical namespace of XML onto the non-hierarchical namespace of SIENA. XML allows items to have duplicate elements, but SIENA does not allow events to contain duplicate attribute names. XML also considers the placement of the element to be an implicit property of the element. If an XML element is shifted in a document, the meaning of the document may change in subtle ways. Since the SIENA namespace is non-hierarchical, shifting the order of events does not alter the underlying

interpretation of the event. Due to this mismatch in strategies, a conflict arises. If this problem is not dealt with, information about the event may be lost.

6 Possible Solutions

This section will examine possible strategies for mapping a hierarchical namespace onto a non-hierarchical event.

6.1 No handling of hierarchical events

The currently implemented solution in the SIENA-XML API is to disregard multiplicity of event attributes. If multiple XML elements would resolve to the same SIENA attribute name, only one of the attribute values will be sent across the SIENA network. This results in loss of event data, and depending upon the event, this loss may or may not be acceptable.

6.2 Change the event source

Another way to resolve this problem is to modify the format of the event source directly. The event source itself can be altered to filter out multiple event attributes before the XML document is generated. However, this requires that the event source lose a degree of flexibility present in the XML document. The event source and its other XML clients can now not take full advantage of XML - therefore, other representations and event services might be better-suited for this event source.

6.3 Modify the event source

Besides changing the XML event source directly, custom XSLTs can be introduced which remove the multiplicity in a predetermined fashion. XSLTs are a mechanism for translating one XML document into another [Cla99]. By introducing a custom XSL, each document can be coalesced individually into a format that does not contain repetition. However, by decoupling the translation from the actual event source, it is possible that the two documents will become out-of-sync especially if the two documents are not maintained by the same source.

6.4 Regular expression and wildcard event support

A potential solution, and the one most commonly implemented in smaller-scale event services, is to use wildcard filtering for attribute names. This allows a client to subscribe with a filter of foo.bar<0-9> or foo.*.bar, and any names matching that criteria will be examined with the covering relations. However, this introduces a burden upon the SIENA server which would severely impact the scalability of the SIENA network. Therefore, regular expression and wildcard support can not be added to the SIENA system.

6.5 Out-of-Band support

Another possible alternative is to develop an automated mechanism for dealing with event naming conflicts outside of the SIENA system that does not depend upon altering the XML document. This approach differs from the others in that no changes to the SIENA architecture is required, and a generic XML-conversion library can still be maintained.

A potential design would be to have an automated daemon listening to specific types of requests on the SIENA network. SIENA applications that were designed to utilize this daemon would contact it when an unreconcilable attribute name is encountered from foreign data source. Before publishing the event, it would notify this daemon of the original names encountered in the document and the new names that it has generated to compensate for the loss of

information. This daemon would record the relevant information in a database. Knowledgeable SIENA applications could then contact the daemon to translate any foreign names to their new automated names.

Knowledge of such a protocol would have to be built into each SIENA client in order for it to succeed. The genesis of such a system originates from DNS[Moc87]. Every computer on the Internet is virtually required to have a DNS resolver for mapping human-readable names to IP addresses. DNS is designed to take inherently hierarchical namespaces and present them in a flat hierarchy (such as computer networks to IP addresses). DNS is also designed to be extremely fault-tolerant and highly distributed - characteristics that such a naming service for SIENA should also possess. However, the cost of developing such a system may make the barrier of entry higher than the other strategies presented here.

7 Conclusion

The SIENA-XML project accomplished what it set out to do - create an end-to-end demonstration of the SIENA system. During the course of this research, a real-world problem was encountered. Mapping from a hierarchical namespace to a non-hierarchical namespace is not insurmountable, but it does require a coherent strategy to solve the problem. This paper has attempted to provide an overview of the relevant SIENA and XML technologies, a brief summation of the class hierarchy used in the current implementation of SIENA-XML, provide details and examples about the mapping problem. Finally, some strategies are suggested for resolving this conflict.

At this point, it is not unclear which mapping strategy would prove to be most successful. As discussed above, each strategy has inherent advantages and disadvantages. Each solution merits further research and exploration into its viability. The solution to such a problem is not necessarily trivial in an event service designed for scalability. Yet, when a proper solution is found, an entirely different class of events will be easily transmitted on the SIENA network.

8 Appendix - Example conversion

8.1 Sample XML event source

The following is a sample XML event source derived from http://www.slashdot.org/slashdot.xml. This example has been trimmed slightly to ease the length requirements.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<backslash xmlns:backslash="http://slashdot.org/backslash.dtd">
 <story>
  <title>Game Boy Advance Arrives</title>
  <url>http://slashdot.org/article.pl?sid=01/03/21/0221200</url>
  <time>2001-03-21 02:50:59</time>
  <author>timothy</author>
  <department>change-is-afoot</department>
  <topic>games</topic>
  <comments>89</comments>
  <section>articles</section>
  <image>topicgames.jpg</image>
 </story>
 <story>
  <title>Forced Into Spamming By Your Employer?</title>
  <url>http://slashdot.org/article.pl?sid=01/03/20/0944212</url>
  <time>2001-03-20 23:37:58</time>
  <author>Cliff</author>
  <department>forced-between-a-rock-and-a-tin-of-canned-meat</department>
  <topic>news</topic>
  <comments>362</comments>
  <section>askslashdot</section>
  <image>topicnews.gif</image>
 </story>
 <story>
  <title>H2G2: Back At Last, With Moderation</title>
  <url>http://slashdot.org/article.pl?sid=01/03/20/2211216</url>
  <time>2001-03-20 23:32:29</time>
  <author>timothy</author>
  <department>no-triple-breasted-whore?</department>
  <topic>news</topic>
  <comments>40</comments>
  <section>articles</section>
  <image>topicnews.gif</image>
 </story>
 <story>
  <title>The Honeynet Project Has A Winner</title>
  <url>http://slashdot.org/article.pl?sid=01/03/20/2127217</url>
  <time>2001-03-20 22:22:04</time>
  <author>timothy</author>
  <department>and-a-loser</department>
  <topic>announce</topic>
  <comments>109</comments>
  <section>articles</section>
  <image>topicannouncements.gif</image>
 </story>
</backslash>
```

8.2 SIENA-XML Rendering

This is one story contained in XML documented referenced in Section 8.1:

backslash.story.title	Game Boy Advance Arrives
backslash.story.url	http://slashdot.org/article.pl?sid=01/03/21/0221200
backslash.story.time	2001-03-21 02:50:59
backslash.story.author	timothy
backslash.story.department	change-is-afoot
backslash.story.topic	games
backslash.story.comments	89
backslash.story.section	articles
backslash.story.image	topicgames.jpg

Here is another event that could be generated from that same data source:

backslash.story.title	Forced Into Spamming By Your Employer?
backslash.story.url	http://slashdot.org/article.pl?sid=01/03/20/0944212
backslash.story.time	2001-03-20 23:37:58
backslash.story.author	Cliff
backslash.story.department	forced-between-a-rock-and-a-tin-of-canned-meat
backslash.story.topic	news
backslash.story.comments	362
backslash.story.section	ask slashdot
backslash.story.image	topicnews.gif

References

[Car98] Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy, December 1998.

[Cla99] James Clark. XSL Transformations. *W3C*, 1999. http://www.w3.org/TR/xslt.

[CRW00] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, Portland OR, USA, July 2000.

[Moc87] P. Mockapetris. Domain names - concepts and facilities. RFC 1034, Network Working Group, ISI, 1987. http://www.rfc-editor.org/rfc/rfc1034.txt.

[XML] XML News. http://www.xmlnews.org/RSS/content.html.