

XML Language Binding Support for Pervasive Communication in Distributed Virtual Shared Information Spaces

Norbert Luttenberger Florian Reuter Jochen Koberstein
Communication Systems Research Group
Institute for Computer Science and Applied Mathematics
Christian-Albrechts-University in Kiel, Germany
[nl|fr|jko]@informatik.uni-kiel.de

Abstract

In this paper we show a novel middleware approach for pervasise applications that communicate via an XML-based distributed virtual shared information space. This approach extends some common ideas on XML language binding frameworks by a dedicated “merge logic” that lets pervasive devices share their information with low overhead.

1. Introduction

Exchanging information via reading from and writing to a shared information space (SIS) is an appropriate communication mechanism for many pervasive applications. Devices may, for instance, announce their presence and their capabilities to other devices by writing to the information space, and learn about the presence of others by reading from it. A big advantage for SIS-mediated exchange of application-specific data obviously results from the fact that no point-to-point routes have to be set up, which can be very costly, especially in mobile environments.

An important concern however is that in pervasive computing environments the installation of something like a centralized “information space server” is not a viable way. In this article, we therefore propose to set up fully distributed virtual Shared Information Spaces (dvSIS). A dvSIS is looked upon as a virtual entity, as it exists as an abstraction only. Every device holds a local instance of the dvSIS, but this instance may be incomplete, partially obsolete, or inconsistent with the local instances of other nodes. That is, we omit the need for a centralized SIS by relaxing the requirements for consistency and coherency of shared information.

To contribute to the dvSIS, every node in the pervasive environment simply broadcasts shared information to the nodes in its vicinity. As we assume that nodes are connected by radio, broadcasting seems to be the “natural” way of message transmission. Nodes may help

each other by relaying data, i.e. by re-broadcasting data received from others. In another paper, we have shown how the amount of data being broadcasted in such environments can be tightly controlled by a class of content-based flooding control algorithms which we call XCast [19].

In this paper, we present a language binding support system for pervasive application software systems that communicate via a dvSIS. It imposes a straightforward methodology on the development process for pervasive applications. It is based upon a schema-typed XML coding of the dvSIS, and we call it STAX, which stands for “Simple Typed API for XML”.

The paper is organized as follows: In the following chapter, we give some reasons why an XML coding for the dvSIS is favourable. We then show the structure of pervasive applications developed with STAX. In chapter 4, we make some remarks on related work, and conclude with an outlook.

2. Schema-Typed XML Coding

Though often criticized for its—sometimes excessive—overhead, we decided to use a schema-typed XML coding for the dvSIS such that the dvSIS can be looked upon as a “common virtual XML document”. The following two arguments lead our decision:

- XML enables the exchange of self-describing information elements between nodes. Messages being broadcast by pervasive nodes can be designed as “tinyfied” dvSIS instances, i.e. as instances that—though possibly carrying minimal content—fit well into the dvSIS schema definition. (The required versatility of the dvSIS schema definition can be preserved, for instance, by appropriate tree structuring with a minOccurs-constraint of zero for most branches.)
- The ability of XML to express semi-structuredness enables the designer to appropriately reflect variations in the configuration of a pervasive computing environment. Local dvSIS instances may

differ in structure or content or both; differences are “legal” unless they are not incompatible with the “global” dvSIS schema.

For definition of the dvSIS schema, we prefer the W3C XML Schema Definition Language, as it is the widest-spread language of this kind.

A middleware supporting the dvSIS paradigm has to support application programs in sending and receiving the above outlined dvSIS instances, and it has to support the application program in merging any received valid messages with the local dvSIS instance. Before sending a “fresh” message or relaying a received message the middleware has to apply the above mentioned flooding control algorithms. For this subject, we again refer to [19], and instead now introduce an application example that will help us to explain how the STAX middleware supports application programs in processing received valid message.

The example can be described as follows: Participants of a conference are equipped with wireless-enabled address books working in ad-hoc mode. Initially, a device holds an address entry for its owner only, consisting of a name field and optionally an e-mail address field. Eventually, every address book application program starts to broadcast the owner address and the address entries that have become known to it. It is the applications’ goal that after some time, all address books hold the addresses of all conference participants. The schema for the related dvSIS is given in Figure 1.

```

<xsd:schema xmlns:xsd=
  "http://www.w3.org/2001/XMLSchema">
<xsd:element name="addressbook"
  type="ABookType"/>
<xsd:complexType name="ABookType">
<xsd:sequence>
<xsd:element name="address"
  type="AddrType"
  maxOccurs="5000"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AddrType">
<xsd:sequence>
<xsd:element name="name"
  type="xsd:string"/>
<xsd:element name="email"
  type="xsd:string"
  minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

Figure 1. Sample XML schema

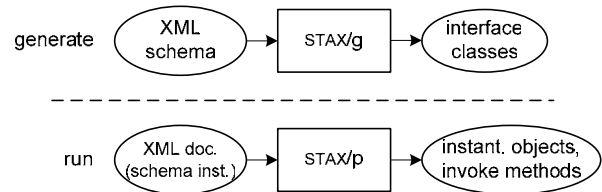


Figure 2. STAX generator and parser

3. STAX-based Applications

3.1. Outline

XML language bindings are “software mechanisms that transform XML data into values that programmers can access and manipulate from within their language of choice” [26]. A language binding framework provides code for deserialization and serialization of schema-typed XML documents.

In the context of the STAX language binding support system, application programs are developed in three steps:

1. A schema is developed specifying the dvSIS.
2. From the schema, a generator called STAX/g (see Figure 2) generates a schema dedicated language binding framework. This framework comprises a number of abstract classes with abstract methods that implement the interface between the dvSIS and the pervasive application logic (see Figure 3). It additionally includes an automaton for a validating parser that we call STAX/p.
3. The programmer “fills in” the application logic into the generated framework’s abstract methods.

At run-time the STAX/p XML parser checks the validity of the incoming XML documents and passes them to the application logic via the generated framework.

3.2. The STAX Concept

The language binding concept applied by STAX can be characterized by three terms: event-based, typed, and patterned after the factory design pattern [13].

The event-based approach was chosen for STAX mainly for two reasons:

- Event-based language binding frameworks are much less resource-consuming than memory-based frameworks, where an XML document is in total stored in memory. Event-based frameworks need much less memory because their only task is to generate events when parsing an XML document. It is up to the application to store, aggregate, discard or do something else with the elements from the parsed document.

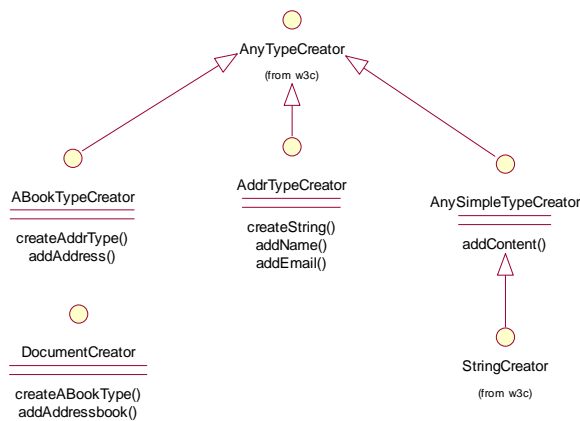


Figure 3. Class hierarchy generated by STAX/g

- Event-based frameworks give the programmer the freedom to store the content of elements from the parsed XML document in any data structure of his or her choice. This may for instance be a single bit, when the XML document holds an element with only two acceptable enumerated values: YES or NO.

A typed framework provides to the programmer programming language objects to handle the information items contained in an XML document, which helps the programmer to stay in his/her world: He or she can concentrate on data presented by objects and does not have to handle DOM's or SAX's element nodes, attribute nodes and text nodes.

The factory design pattern (belonging to the creational pattern category) was godfather when deciding that STAX/g should generate a class per complexType, simpleType, Group, and attributeGroup showing up in the dvSIS schema.

In the next section, we are going to show how STAX is used in a “basic mode”, i.e. without any concern for shared information spaces; then we introduce a “merge logic” that makes the STAX approach usable for applications that communicate via a dvSIS.

3.3. “Basic mode” STAX

When generating a language binding framework, STAX/g maps the schema type hierarchy into the class hierarchy of an object-oriented programming language by “mirroring” all schema-defined types by so-called creator classes. (To enable this kind of mapping, a STAX/g pre-processor transforms all anonymous types eventually showing up in a schema into named types.) In the example the AddrType type as defined in the schema corresponds to the AddrTypeCreator class (Figure 3). In

order to maintain the programming language type hierarchy, the anyType type is the root of all types.

STAX/g equips any parent creator class with create and add methods for its child classes. The create method allows to instantiate an object of the child class, the add method allows to add this object to the parent object. If an element's type is a simpleType (in terms the W3C XML Schema definition language)—or in other words: if the element's content is character data—, then STAX/g adds an addContent method to the related class. This method allows to pass the element's content to the object.

Due to space limitations, we cannot describe further details, especially the handling of attributes and special schema constructs like groups or unions. STAX/g has been designed and implemented such that it can process all the constructs as foreseen in the W3C XML Schema definition language.

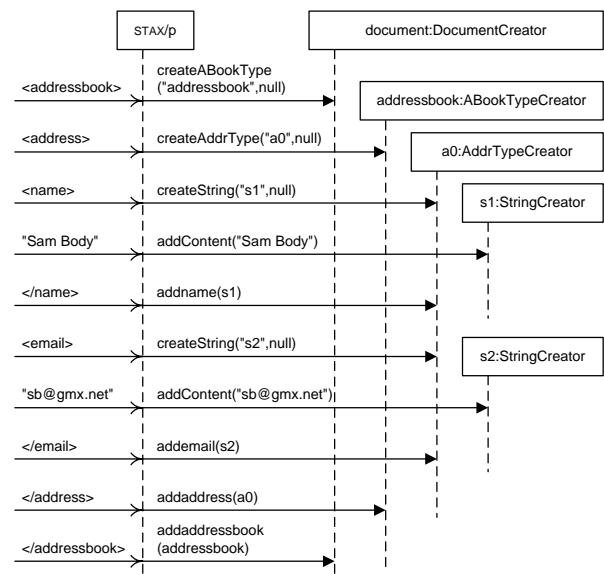


Figure 4. Basic sequence diagram

To illustrate this complex relation between the XML world and the object-oriented world, Figure 4 shows an UML sequence diagram when STAX/p analyses the sample document given in Figure 5.

At runtime—i.e. when parsing a schema instance (i.e. a document)—STAX/p invokes the mentioned methods. When STAX/p reads an element's start tag, it invokes the create method dedicated to the element's type. STAX/p invokes the dedicated add method when reading an element's end tag, which lets the parent object accept the (transformed) content of a child element. The name of an add method is formed from the element name as given in the schema and the prefix add: in our example the add method for the Address element is called addAddress. This method belongs to the ABookCreator class. As the

addContent method is invoked in the context of a typed object, the programmer can transform the passed string to an internal representation in a specific manner. The documentCreator class shown in Figure 3 is instantiated by STAX/p automatically when starting the parsing process.

```
<addressbook>
  <address>
    <name>Sam Body</name>
    <email>sb@gmx.net</email>
  </address>
</addressbook>
```

Figure 5. Sample schema instance

3.4. STAX for dvSIS applications

The schema presented in Figure 1 allows an electronic addressbook to broadcast XML documents that contain its owner's name only, but not its e-mail address. This is because the <email> particle of the <address> element has an occurrence cardinality minOccurs="0". (By default, elements/particles have occurrence cardinalities minOccurs="1" and maxOccurs="1".) Now, we can imagine a situation where—after some time—the addressbook owner adds an additional e-mail address to his or her addressbook entry. Obviously, this must not produce a new entry in the addressbooks of those conference participants that have received and stored the incomplete entry before. Instead, the "old" entry must be corrected, i.e. merged with the new information. For this purpose, the STAX framework has been enhanced by two additional methods: the contains and the merge methods.

```
<addressbook>
  <address>
    <name>Sam Body</name>
    <e-mail>sambo@web.de</e-mail>
  </address>
</addressbook>
```

Figure 6. Another sample entry

In the following we describe something like a "default implementation" for these methods. Extensions are possible, allowing application-specific delete and rewrite operations which may e.g. take into account context information.

Merging starts after invocation of the above described add method. The add method returns true, if the information to be accepted was previously unknown, and false otherwise. When returning false, the add operation actually has not been carried out. In reaction to a false

return value, the STAX/p parser invokes the contains method. This method returns the instance of the class, which was added by a previous invocation of the add method. (It would return null if none such instance existed.) STAX/p now invokes the merge method instead of the add method. The merge method takes the passed object and merges it with the indicated object.

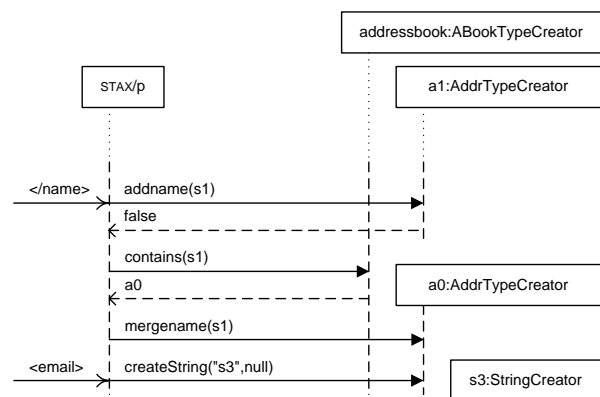


Figure 7. Merge mode sequence diagram

For merging, STAX implicitly defines keys. In our example, the <name> element obviously is a key for the addrtype type. In general, it is valid that an element must be identifiable before any information can be merged into this element. An element is said to be identifiable if all of its particles that have occurrence cardinalities minOccurs="1" and maxOccurs="1" have been added to the element.

The sequence diagram fragment of Figure 7 illustrates how an incoming XML document is successfully merged with the local instance of an addressbook.

Obviously, the schema design is critical for the function of the merge algorithm: If, for instance, a merge of instance a0 in Figure 4 with another entry as given in Figure 6 would result in an instance with two e-mail addresses, which is not valid regarding the schema in Figure 1. A better design for the schema would be to set the maxOccurs occurrence property of the e-mail particle to unbounded. In this case the merge would comply with the schema.

It should not be neglected that the runtime of the merge algorithm is critically influenced by the schema design. If, for instance, the schema comprises one object which makes it identifiable and optionally 100 data values, it would be a good idea to send the identification object first—and then the optional data values, because the merge operation is much faster then.

4. Related Work

An alternative way to instantiate and maintain a dvSIS would be the usage of “classical” middleware techniques and systems like e.g. [1,5,6,9,11,14,15,16,22,25,31]. These systems have underlying cooperation paradigms similar to the publisher/subscriber paradigm, and provide supporting services like routing and route maintenance in ad-hoc networks [23] for access to one or more XML-based tuple space-like servers [7,21,30]. Tuple space-like servers centrally store shared information and provide data to their clients through middleware services. The STAX-based middleware approach differs fundamentally from these approaches in a way, that the STAX-based middleware does not require any routing messages, beacons and management packets and thus reduces overhead considerably.

XML merging algorithms have been studied in the context of XML data bases [17,29], as an equivalent to the join operation known from relational data bases. These algorithms require special data structures and guidance with keys and XPath expressions. Other XML merging algorithms were developed for version management systems [20]. Here it is the idea that two versions of a document are modified independently by different people, and one needs to merge them. The process is semi-automatically, i.e. the algorithm may require human guidance.

The concept of language binding is very new and is currently used in products like .NET, J2EE, Castor and others [2]. These products are designed for current personal computers and application servers and are quite resource consuming. An resource-sparing language binding which is integrated in the presented middleware has been especially developed within the STAX project parallel with the presented middleware.

5. Outlook

Currently, we are implementing STAX versions for chipcards and for the C programming language. The latter are aiming at very small devices which are mostly programmed in C.

At the same time, some simulations are conducted to study the scalability of the mentioned XCast flooding control algorithm.

References

- [1] UPnP Forum. <http://www.upnp.org/>.
- [2] XML data binding resources. <http://www.rpbouret.com/xml/XMLDataBinding.htm>.
- [3] S. Al-Khalifa, H. V. Jagadish, N. Koudas, and J. M. Patel. “Structural Joins: A Primitive for Efficient XML Query Pattern

Matching”, <http://www.eecs.umich.edu/jignesh/publ/xmljoin-ICDE.pdf>.

- [4] M. Altinel, M.J. Franklin. “Efficient Filtering of XML Document for Selective Dissemination of Information”, *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 2000.
- [5] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. “Challenges: An Application Model for Pervasive Computing”, *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.
- [6] P. Bonnet, J. Gehrke, and P. Seshardri. “Querying the Physical World”, *IEEE Personal Communications* 7(5), Oct. 2000, pp.10–15.
- [7] G. Cabri, L. Leonardi, and F. Zambonelli. “XML Dataspaces for the Coordination of Internet Agents”, *Applied Artificial Intelligence* 15(1), Jan. 2001, pp. 35–58.
- [8] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. “Achieving Scalability and Expressiveness in an Internet-scale Event Notification Service”, *Symposium on Principles of Distributed Computing 2000*, pp. 219–227.
- [9] P. Castro, R. Muntz. “Managing Context Data for Smart Spaces”, *IEEE Personal Communications* 7(5), Oct. 2000, pp. 44–46.
- [10] M. Cherniack, M. J. Franklin, and St. Zdonik. “Expressing User Profiles for Data Recharging”, *IEEE Personal Communications* 8(4), Aug. 2001, pp.32–38 .
- [11] L.M. Feeney, B. Ahlgren, and A. Westerlund. “Spontaneous Networking: An Application-oriented Approach to Ad-hoc Networking”, *IEEE Communications Magazine* 39(6), June 2001, pp. 176–181.
- [12] R. La Fontaine. “Merging XML Files: A New Approach Providing Intelligent Merge of XML Data Sets.” *XML Europe 2002*, May 2002, <http://www.idealliance.org/papers/xml02/dxxmle02/papers/03-03-04/03-03-04.pdf>.
- [13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Publishing, 1996.
- [14] St.D. Gribble, M. Welsh, J.R. von Behren, E.A. Brewer, D.E. Culler, N. Borisov, St.E. Czerwinski, R. Gummadi, J.R. Hill, A.D. Joseph, R.H. Katz, Z.M. Mao, S. Ross, and B. Y. Zhao. “The Ninja Architecture for Robust Internet-scale Systems and Services”, *Computer Networks* 35(4), 2001, pp. 473–497.
- [15] Ch. Herring, S. Kaplan. “Component-based software Systems for Smart Environments”, *IEEE Personal Communications* 7(5), Oct. 2000, pp. 60–61.
- [16] T. Imielinski, S. Goel. “DataSpace: Querying and Monitoring Deeply Networked Collections in Physical Space”, *IEEE Personal Communications* 7(5), Oct. 2000, pp. 4–9.
- [17] H. Jagadish, L. Lakshmanan, D. Srivastava, and K. Thompson. “Tax: A Tree Algebra for XML”, <http://citeseer.nj.nec.com/jagadish01tax.html>.
- [18] J. Kaiser, M. Mock. “Implementing the Real-time Publisher/subscriber Model on the Controller Area Network (CAN)”, *Proceedings of the Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing 1999*.
- [19] F. Reuter, J. Koberstein, and N. Luttenberger. “The XCast Approach for Content-based Flooding Control in Distributed Virtual Shared Information Spaces – Design and Evaluation”,

Praxis der Informationsverarbeitung und Kommunikation 26(4), Oct. 2003, pp. 216–222.

[20] G.W. Manger. “A Generic Algorithm for Merging SGML/XML Instances”, *XML Europe 2001*, May 2001. <http://www.gca.org/papers/xml europe2001/papers/html/s29-1.html>.

[21] D. Moat. “XML-Tuples and XML-Spaces”, <http://uncled.oit.unc.edu/XML/XMLSpaces.html>.

[22] M. Nidd. “Service Discovery in DEAPspace”, *IEEE Personal Communications* 8(4), Aug. 2001, pp. 39–45.

[23] Ch.E. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2001.

[24] L. Seligman, A. Rosenthal. “XML’s Impact on Databases and Data Sharing”, *IEEE Computer Magazine* 34(6), June 2001, pp. 59–67.

[25] Ch. Shen, Ch. Srisathapornphat, and Ch. Jaikkaeo. “Sensor Information, Networking Architecture and Applications”, *IEEE Personal Communications* 8(4), Aug. 2001, pp. 52–59.

[26] Simeoni, Manghi, Lievens, Connor, and Neely. “An Approach to High-level Language Bindings to XML”, *Information and Software Technology* 44(4), March 2002, pp. 217–228.

[27] H. Su, H. Kuno, and E.A. Rundensteiner. “Automating the transformation of XML documents”, *Proceedings of the Third International Workshop on Web Information and Data Management*, Atlanta, Georgia, USA, 2001.

[28] K. Tufte, D. Mater. “Aggregation and accumulation of XML Data”, *IEEE Data Engineering Bulletin* 24(2), 2001, pp. 34–39.

[29] K. Tufte, D. Maier. “Merge as a Lattice-join of XML Documents”, <http://www.cs.wisc.edu/niagara/papers/paper608.pdf>.

[30] P. Wycko. “T spaces”, *IBM Systems Journal* 37(3), 1998, <http://www.research.ibm.com/journal/sj/373/wycko.html>.

[31] B.Y. Zhao, A. Joseph. “The XSet search engine and Xbench XML query benchmark”, Technical Report UCB/CSD-00-1112, Sept. 2000, Computer Science Division (EECS), University of California, Berkeley, California 94720.