XMLSpaces for Coordination in Web-based Systems

Robert Tolksdorf Dirk Glaubitz

Technische Universität Berlin, Fachbereich Informatik, FLP/KIT,
Sekr. FR 6–10, Franklinstr. 28/29, D-10587 Berlin, Germany,
mailto:tolk@cs.tu-berlin.de, *http://www.cs.tu-berlin.de/˜tolk*, *mailto:glaubitz@cs.tu-berlin.de*

February 23, 2001

Abstract

XMLSpaces is an extension to the Linda coordination language for Web-based applications. It supports XML documents as tuple fields and multiple matching routines implementing different relations amongst XML documents, including those given by XML query-languages. XMLSpaces is distributed with a clearly encapsulated open distribution strategy.

1 Introduction

While the Web has become *the* universal information system worldwide in its first ten years of existence, the progress towards open distributed applications utilizing the Web for universal access is rather slow. Although there are several technologies like Java or CORBA available, none of these has reached universal acceptance. The current trend towards application service provision also does not allow for real peer-to-peer distribution, but introduces a server-centric paradigm with very limited distribution.

A core question in supporting distributed applications is what concept is applied for the coordination of independent activities in a cooperative whole. This has been the subject of the study of coordination models, languages and systems ([9]). One approach is to design a separate *coordination language* ([7]) that deals exclusively with the aspects of the interplay of entities and provides concepts orthogonal to computation.

Most recently, the Web-Standard XML (*Extensible Markup Language*) ([14]) has become *the* format to exchange data markup following application specific syntaxis. It may well be the dominating interchange format for data over networks for the next years. XML data is semi-structured and typed. A DTD (*Document Type Definition*) defines a context-free grammar to which an XML document must adhere. Tags define structures within a document that encapsulate further data. With attributes, certain meta information about the data encapsulated can be expressed. While XML enables collaboration in distributed and open systems by providing common data formats, it is still unclear how components coordinate their work.

The concept of *XMLSpaces* presented in this paper marries the common communication format XML with the coordination language Linda. It aims at providing coordination in open distributed systems based on Web-related standards. XMLSpaces provides a simple yet flexible concept to coordinate components in that context and extends the original Linda-notion with a more flexible matching concept.

This paper is organized as follows. We first motivate the extension of Linda with support for XML documents. Then we describe the concepts used in XMLSpaces for multiple XML matching relations and distribution. After that, we describe the components used in the XMLSpaces implementation and finally compare our approach with related work and conclude.

2 Linda-like Coordination

Linda-like languages are based on data-centric coordination models. They introduce the notion of a shared dataspace that decouples partners in communication and collaboration both in space and time ([3]).

The coordination media in Linda is the *tuplespace* which is a multiset of *tuples*. These are in turn ordered lists of unnamed fields typed by a set of primitive types. An example is $\langle$10,"Hello"$\rangle$ which consists of an integer and a string.

The tuplespace provides operations that uncouple the coordinated entities in time and space by indirect, anonymous, undirected and asynchronous communication and synchronization. The producer of data, can emit a tuple to the tuplespace by *out*($\langle$*10,"Hello"*$\rangle$). The consumer of that data does not even have to exist at the time it is stored in the space. The producer can terminate before the data is consumed.

To consume some data, a process has to describe what kind of tuple shall be retrieved. This description is called a *template*, which is similar to tuples with the exception, that fields also can contain bottom-elements for each type, eg. $\langle 10, ?string \rangle$. These placeholders are called *formals* in contrast to *actuals* which are fields with a value. Given a template, the tuplespace is searched for a *matching* tuple. A *matching relation* on templates and tuples guides that selection.

Retrieving a matching tuple is done by *in*($\langle 10, ?string \rangle$) which returns the match and removes it from the space. The primitive *rd*($\langle 10, ?string \rangle$) also returns a match but leaves the tuple in the tuplespace. Both primitives *block* until a matching tuple is found, thereby synchronizing the consumer with the production of data.

In Linda, the matching relation requires the same length of tuples and templates and identical types of the respective fields. For formals in the template, the actual in the tuple has to be of same type, while actuals in the template require the same value in the tuple.

Tuples as in Linda can be considered "primitive data" – there are no higher order values such as nested tuples, no mechanisms to express the intention of typing fields such as names etc.

When aiming at coordination in open distributed systems, a richer form of data is needed. It has to be able to capture application specific higher data-structures easily without the need to encode them into primitive fields. The format has to be open so that new types of data can be specified. And it has to be standardized in some way, so that data-items can be exchanged between entities that have different design-origins.

The *Extensible Markup Language* XML ([14]) has recently been defined as a basis for application specific markup for networked documents. It seems to meet all the outlined requirements as a data-representation format to be used in a Linda-like system for open distributed systems. *XMLSpaces* is our system that uses XML documents in addition to ordinary tuple fields to coordinate entities with the Linda-primitives.

3 XMLSpaces

XMLSpaces extends the Linda model in several major aspects:

1. XML documents serve as field-data within the coordination space. Thus, ordinary tuples are supported, while pure XML documents can be represented as one-fielded tuples.

2. A multitude of relations amongst XML documents can be used for matching. While some are supplied, the system is open for extension with further relations.

3. XMLSpaces is distributed so that multiple dataspace servers at different locations form one logic dataspace. A clearly separated distribution policy can easily be tailored to different network restrictions.

4. Distributed events are supported so that clients can be notified when a tuple is added or removed somewhere in the dataspace.

We describe these extensions in the following sections.

3.1 XML support in tuple fields

In XMLSpaces, actual tuple fields can contain an XML document, formal fields can contain some XML document description, such as a query in an XML query language. The matching relation is extended on the field-field level with relations on XML documents and expressions from query languages. All Linda operations, and the matching rule for other field-types and tuples are unchanged.

The matching rule to use for XML fields is not statically defined, instead, XMLSpaces supports multiple matching relations on XML documents. The current implementation of XMLSpaces builds on a standard implementation of Linda, namely TSpaces ([17]). It already provides the necessary storage management and the basic implementations for the Linda primitives.

In TSpaces, tuple fields are instances of the class *Field*. It provides a method *matches(Field f)* that implements the matching-relation amongst fields and returns *true* if it holds. The method is called by the matching method of class *SuperTuple*, which tests for equal length of tuples and templates. Actuals and formals are not modeled as distinguished classes but rather typed according to their use in matching.

XMLSpaces introduces the class *XMLDocField* as a subclass of *Field*. The contents of the field is *typed* as an actual or a formal by fulfilling a Java-interface. If it implements the interface *org.w3c.dom.Document*, it is an actual field containing an XML document. If it implements the interface *XMLMatchable*, it is a formal. Otherwise it is an invalid contents for an *XMLDocField*.

The method *matches* of an *XMLDocField* object tests the polarity of fields for matching. It returns false, when both objects are typed as formals, or when an actual is to be matched against a formal. If both the *XMLDocField*-object and the parameter to *matches* are actuals, a test for equality is performed. Otherwise – if a formal is to be matched against an XML document – the method *xmlMatch* of the formal is used to test a matching relation.

Figure 1 summarizes the resulting class hierarchy.

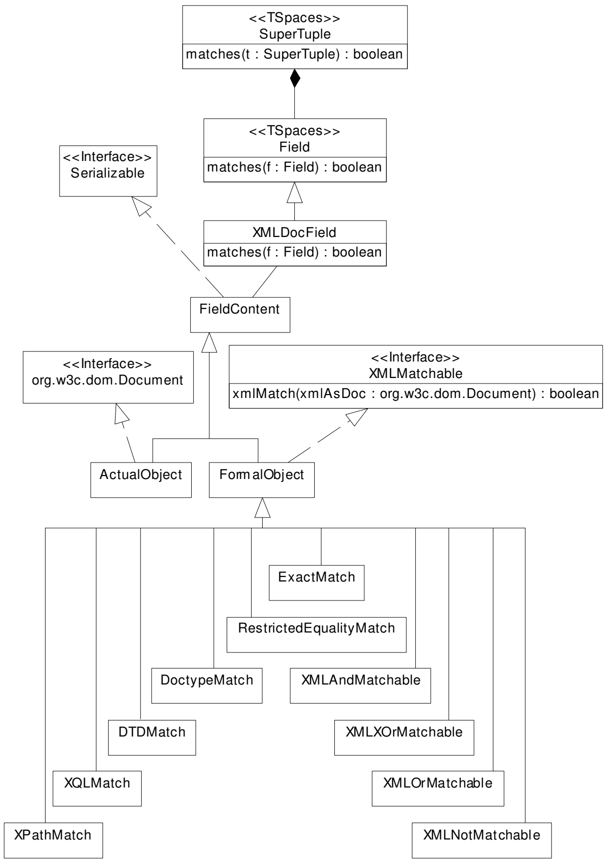


Figure 1. The class hierarchy for XML documents in tuple fields

3.2 Multiple matching relations

The purpose of the interface *XMLMatchable* is to allow for a variety of matching relations amongst XML documents. The template used for *in* and *rd* then, is not relative to the language definition as with Linda, but relative to a relation on XML documents and XML templates that is contained within the template as the implementation of *xmlMatch* in *XMLMatchable*.

The use of multiple matching relations can be an application requirement. In the Workspaces architecture ([11, 12]), workflows and their steps, and the documents worked on are represented as XML documents. In order to retrieve "*something* to do", one wants a document that follows some work-description DTD. If, however, a specific task is to be done on a specific document, one wants the *one* XML document that might be described by an identifier in an attribute. This requirement induces the need for support of multiple relations used in matching.

XMLMatchable is also the basis of the extensibility of XMLSpaces with new matching relations. To realize it, a new class has to be provided that implements this interface and tests for the new relation in the *xmlMatch* method.

While the XML standard defines one relation, namely the "validates" from an XML document to a DTD, there is a variety of possible other relations amongst XML documents and other forms of templates. These include:

- An XML document can be matched to another one based on equality of contents, or on equality of attributes in elements.
- An XML document can be matched to another one which validates against the same grammar, ie. DTD.
- An XML document can be matched to another one which validates against the same minimal grammar with or without renaming of element- and attribute-names.
- An XML document can be matched to a query expression following the syntax and semantics of those, for example XML-QL, XQL, or XPath/Pointer.

Currently, several relations are implemented in XMLSpaces as shown in table 1.

The relations fall into different categories:

- The *equality* relations use several views on what equality of XML documents actually means. For example, are comments included in a check or not.
- The *DTD* relations take the relation of a document to a DTD or a doctype name as constituent for matching.
- The *query language* relations build on several existing XML oriented query languages. A query describes a set of XML documents to which the query matches. The query match then is take as the matching relation in the sense of XMLSpaces. Note that the query languages are of high expressibility, for example, matching for all documents that contain a specific value in some attribute can be formulates as an XPath or XQL expression. While the equality and DTD relations consider a document as a whole, the query relations try to find a match in one part of a document.
- The *connector* relations allow it to build boolean expressions on matching relations whose result gives the final matching relation.

Relation	Meaning	Tool used
Exact equality	Exact textual equality	DOM interfaces
Restricted equality	Textual equality ignoring comments, processing instructions, etc.	DOM interfaces
DTD	Valid towards a DTD	IBM XML4J Parser
DOCTYPE	Uses specific Doctype name	DOM DocumentType
XPath	Fulfills an XPath expression	Xalan-Java
XQL	Fulfills an XQL expression	GMD-IPSI XQL-Engine
AND	Fulfills two matching relations	–
NOT	Does not fulfill matching relation	–
OR	Fulfills one or two matching relations	–
XOR	Fulfills one matching relation	–

Table 1. Matching relations in XMLSpaces

3.3 Distributed XMLSpaces

In order to make XMLSpaces usable to coordinate wide-area applications, it has to support some form of distribution. It seems to be without question, that centralized coordination platforms suffer from major problems concerning performance and communication bottlenecks, single point of failure etc. XMLSpaces supports the integration of XMLSpaces servers at different places into a single logic dataspace.

Distribution of a Linda-like system can be implemented using different distribution schema which have different efficiency characteristics:

- *Centralized*: One server holds the complete dataspace.
- *Distributed*: All servers hold distinct subsets of the complete dataspace.
- *Full replication*: All servers hold consistent copies of the complete dataspace.
- *Partial replication*: Subsets of servers hold consistent copies of subsets of the dataspace ([5]).
- *Hashing*: Matching tuples and templates are stored at the same server selected by some hashing function ([1]).

XMLSpaces does not prescribe one specific strategy but encapsulates the distribution strategy applied in a *distributor object*. At its interface, it offers distributed versions of the Linda-primitives. The implementation of the distributor object implements a distribution strategy by the respective versions of these methods.

XMLSpaces is *open* in the sense that, with a suited distribution strategy, servers can join and leave at any time. As the distributor object has to know about registered servers, its interface includes respective methods to register and deregister remote servers.

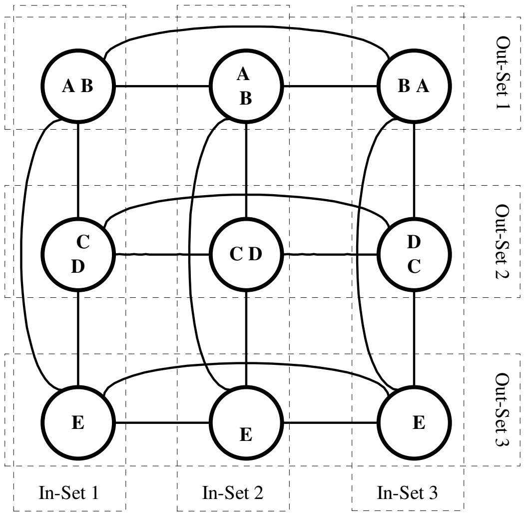
Currently, XMLSpaces includes implementations of the centralized and partial replication strategy. The system can easily be extended by other distributor objects that implement other strategies. The choice of the distribution policy is configured at startup in a configuration file.

The partial replication schema is depicted in figure 2. The nodes in the distributed XMLSpace each store a subset of the complete contents of the datastore. The nodes organized in so called *out-sets* contain identical replicas of a subset as in figure 2(a). An *out*-operation transmits the argument data to all members of the out-set for storage. Every node is at the same time a member of a so called *in-set*. The nodes within one in-set store different subsets of the complete dataspace. The union of their contents represents the whole dataspace. Thus, any *in*- or *rd*-operation asks all nodes in the in-set for a match. In contrast to [10], XMLSpaces does not use a software-bus for communication but establishes point-to-point communication amongst the members of a set.

The structure formed by the sets has to be rectangular – a condition that cannot be upheld in the case of open systems with a varying number of participating nodes. Therefore, the structure has to be retained by *simulating* nodes if necessary. As shown in figure 2(b), one physical node then is part of two out- or in-sets. The reconfiguration of the system in the case of joining and leaving nodes is part of the protocol for joining and leaving nodes.

The distributed strategy above turns out to be a special case of partial replication – one with only one in-set. XMLSpaces offers a respective subclass for distributor objects. Similar, full replication is also the special case of partial replication with a single out-set.

TSpaces supports *events* that can be raised when a tuple is entered or removed from the dataspace. XMLSpaces extends this mechanism to support *distributed events* where clients can register for an event occuring somewhere in the distributed dataspace. The distribution of events is also performed by the distributor object following the respective



(a) The grid

(b) Simulating nodes

Figure 2. Partial Replication

distribution strategy.

4 Implementation

XMLSpaces extends the original Linda conception with XML documents and distribution. It does not change the set of primitives supported nor affect the implemented internal organization of the dataspace. Thus, we have chosen to build on an existing Linda-implementation, namely TSpaces ([17]).

TSpaces is attractive for this purpose, as it is an object-oriented implementation in Java and the XML support can be easily introduced by subclassing. Also, all issues of server management can be reused for XMLSpaces. In order to support distribution, the original TSpaces implementation had to be extended at some places. TSpaces allowed for a rapid implementation of XMLSpaces focusing on the extensions. However, it could well be exchanged by some other extensible Linda-kernel.

The standard document object model DOM ([13]), level 1, serves as the internal representation of XML documents in actual fields. This lead to a great flexibility to extend XMLSpaces with further matching relations using a standard API. It has shown that the integration of such an engine into XMLSpaces is extremely simple when written in Java and utilizing DOM. If not, some wrapper-object has to be specified in addition. XMLSpaces itself is completely generic towards how the *xmlMatch*-method is implemented and what its semantics are.

As seen in table 1, the huge amount of XML related software provided engines that could directly evaluate the relations on XML documents we are interested in.

5 Related Work

There are some projects documented on extending Linda-like systems with XML documents. However, XMLSpaces seems to be unique in its support for multiple matching relations and its extensibility.

MARS-X ([2]) is an implementation of an extended JavaSpaces ([6]) interface. Here, tuples are represented as Java-objects where instance variables correspond to tuple fields. Such an tuple-object can be externally represented as an element within an XML document. Its representation has to adhere to a tuple-specific DTD. MARS-X is closely related to tuples as Java objects and does not look at arbitrary relations amongst XML documents.

XSet ([18]) is an XML database which also incorporates a special matching relation amongst XML documents. Here, queries are XML documents themselves and match any other XML document whose tag structure is a strict superset of that of the query. It should be simple to extend XMLSpaces with this engine.

The note in [8] describes a preversion for an XMLSpaces. However, it provides merely an XML based encoding of tuples and Linda-operations with no significant extension. Apparently, the proposed project was not finished up to now.

TSpaces has some XML support built in ([17]). Here, tuple fields can contain XML documents which are DOM-objects generated from strings. The *scan*-operation provided by TSpaces can take an XQL query and returns all tuples that contain a field with an XML document in which one or more nodes match the XQL query. This ignores the field structure and does not follow the original Linda definition of the matching relation. Also, there is no flexibility to support further relations on XML documents.

6 Conclusion and Outlook

XMLSpaces is a distributed coordination platform that extends the Linda coordination language with the ability to carry XML documents in tuple fields. It is able to support multiple matching relations on XML documents. Both the set of matching relations and the distribution strategy are extensible.

XMLSpaces satisfies the need for better structured coordination data in the Web context by using XML in an open end extensible manner. It has shown that the Linda concept can be extended easily while retaining the original concepts on coordination and a very small core of the coordination language.

Future technological extensions of XMLSpaces include the use of DOM level 2 object model ([15]) to represent XML documents. This standard supports XML Namespaces ([16]) which is necessary to support the full set of XML core specifications in XMLSpaces. Also, this might lead to further matching relations. Issues for extending the functionality are in the areas of security, and fault-tolerance, including extending the transaction concept already existing in TSpaces.

Currently, XMLSpaces is static in its configuration of the distribution policy. A future extension will be support for runtime composition of the system similar to OpenSpaces ([4]). In order to do so, the distributor objects have to be able to establish some "normalized" distribution state from which a new strategy can be built.

Further details about XMLSpaces can be found at URL *http://www.cs.tu-berlin.de/˜tolk/xmlspaces*.

Acknowledgement

The IBM Almaden Research Center supported the work on XMLSpaces by granting a licence to the TSpaces source code.

References

[1] R. Bjornson. *Linda on Distributed Memory Multiprocessors*. PhD thesis, Yale University Department of Computer Science, 1992. Technical Report 931.

[2] G. Cabri, L. Leonardi, and F. Zambonelli. XML Dataspaces for Mobile Agent Coordination. In *15th ACM Symposium on Applied Computing*, pages 181–188, 2000.

[3] N. Carriero and D. Gelernter. Linda in Context. *Commun. ACM*, 32(4):444–458, 1989.

[4] S. Ducasse, T. Hofmann, and O. Nierstrasz. OpenSpaces: An Object-Oriented Framework For Reconfigurable Coordination Spaces. In A. Porto and G.-C. Roman, editors, *Coordination Languages and Models*, LNCS 1906, pages 1–19, Limassol, Cyprus, Sept. 2000.

[5] C. Fraasen. Intermediate Uniformly Distributed Tuple Space on Transputer Meshes. In J. Banâtre and D. Le Métayer, editors, *Research Directions in High-Level Parallel Programming Languages*, number 574 in LNCS, pages 157–173. Springer, 1991.

[6] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces principles, patterns, and practice*. Addison-Wesley, Reading, MA, USA, 1999.

[7] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Commun. ACM*, 35(2):97–107, 1992.

[8] D. Moffat. XML-Tuples and XML-Spaces, V0.7. http://uncled.oit.unc.edu/XML/XMLSpaces.html, Mar 1999.

[9] G. Papadopoulos and F. Arbab. Coordination models and languages. In *Advances in Computers*, volume 46: The Engineering of Large Systems. Academic Press, 1998.

[10] R. Tolksdorf. Laura - A Service-Based Coordination Language. *Science of Computer Programming, Special issue on Coordination Models, Languages, and Applications*, 1998.

[11] R. Tolksdorf. Coordinating Work on the Web with Workspaces. In *Proceedings of the IEEE Ninth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises WET ICE 2000*. IEEE Computer Society, Press, 2000.

[12] R. Tolksdorf. Coordination Technology for Workflows on the Web: Workspaces. In *Proceedings of the Fourth International Conference on Coordination Models and Languages COORDINATION 2000*, LNCS. Springer-Verlag, 2000.

[13] World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, 1998. http://www.w3.org/TR/REC-DOM-Level-1.

[14] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C Recommendation, 1998. http://www.w3.org/TR/REC-xml.

[15] World Wide Web Consortium. Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation, 2000. http://www.w3.org/TR/DOM-Level-2-Core.

[16] World Wide Web Consortium. Namespaces in XML. W3C Recommendation, 2000. http://www.w3.org/TR/REC-xml-names.

[17] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[18] B. Y. Zhao and A. Joseph. The XSet XML Search Engine and XBench XML Query Benchmark. Technical Report UCB/CSD-00-1112, Computer Science Division (EECS), University of California, Berkeley, 2000. September.