Coordinating Web-based Systems with Documents in XMLSpaces

Robert Tolksdorf[1] and Dirk Glaubitz[2]

[1] 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*,
[2] *mailto:glaubitz@cs.tu-berlin.de*

Abstract. We describe an extension to the Linda model of coordination for Web-based applications. It allows XML documents to be stored in a coordination space from where they can be retrieved based on multiple matching relations amongst XML documents, including those given by XML query-languages. XMLSpaces is distributed and supports several distribution policies in an extensible manner. We describe the partial replication schema implemented in detail.

1 Introduction

While the Web has become *the* universal information system worldwide in the first decade of its existence, the progress towards cooperative information systems 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.

A core question in supporting such distributed systems is on the concept 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 ([PA98]). The work described here follows the approach to design a separate *coordination language* ([GC92]) 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*) ([Wor98b]) 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 by an external or internal document type or by a minimal grammar inferred from the given document. 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 ([Gla00,TG01]). It aims at providing coordination in Web-based cooperative information systems. XMLSpaces offers a simple yet flexible approach 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 look at the coordination language Linda and show how XMLSpaces extends it with XML documents in tuple-fields. We describe the extensible support for multiple matching relations amongst XML documents. After that, we describe the distribution concept applied, with special emphasis on the partial replication scheme implemented, and how distributed events are provided. After brief looks at our implementation platform and related work, we 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 ([CG89]).

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 ⟨10,"Hello"⟩ 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 with the operation *out(⟨10,"Hello"⟩)*. 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. ⟨10,?string⟩. 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(⟨10,?string⟩)* which returns the match and removes it from the space. The primitive *rd(⟨10,?string⟩)* 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.

The Linda 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 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 Web-based 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 ([Wor98b]) 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 fields within the coordination space. Thus, ordinary tuples are supported, while 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 this and the following sections.

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 XML 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 ([WMLF98]). 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 called *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 shows the resulting class hierarchy.

4 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

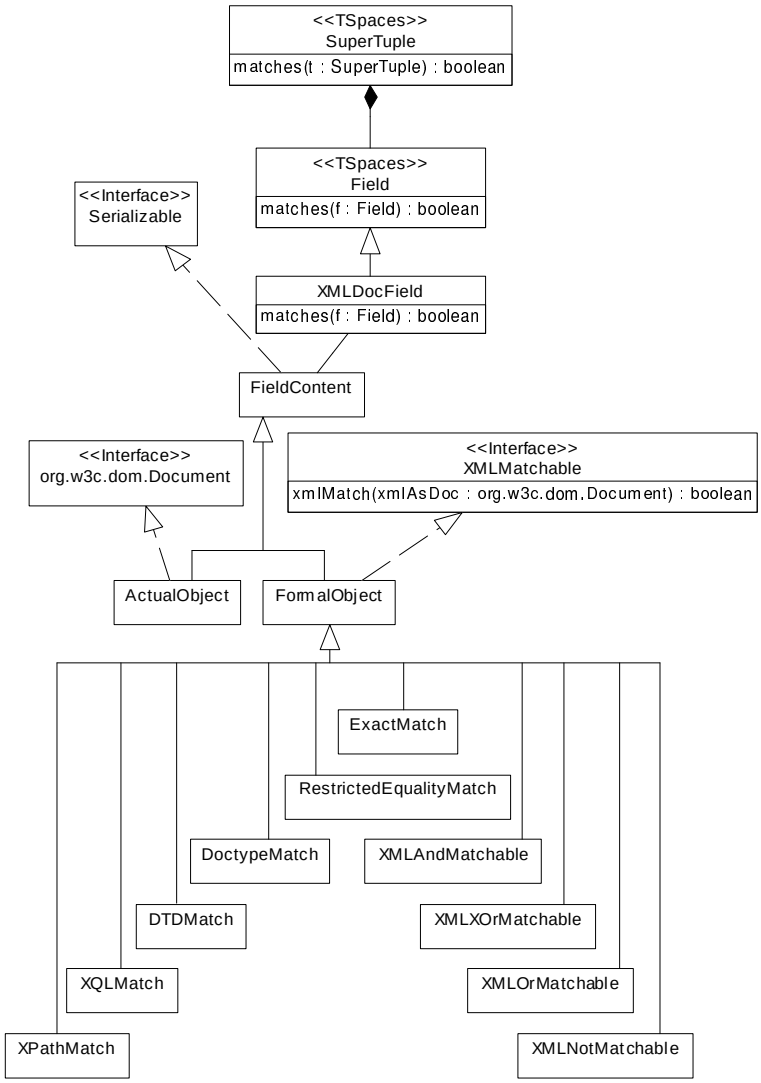


Fig. 1. The class hierarchy for XML documents in tuple fields in UML notation

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 *xml-Match* in *XMLMatchable.*

The use of multiple matching relations can be an application requirement. We find such a requirement in the Workspaces architecture ([Tol00a,Tol00b,TS01]). Workspaces is a Web-based workflow system which combines concepts from the application domain workflow management with standard Internet technology, namely XML and XSL, and with coordination technology.

Steps are the basic kind of activity in Workspaces and represent a unit of work on some application specific XML document. Each step is represented as an XML document that can be distributed individually for interpretation by the XSL-based Workspaces engine. As a result, distributed workflows can be coordinated via the Web. A complete Workflow is described as a graph in an XML-document. It is split into a set of individual steps in an XSL-based compilation step.

Figure 2 shows an example of such a workflow graph – in this case describing the review process for papers submitted to a conference. The application specific documents being manipulated are the papers submitted and reviews forms to be filled out by members of a program committee.

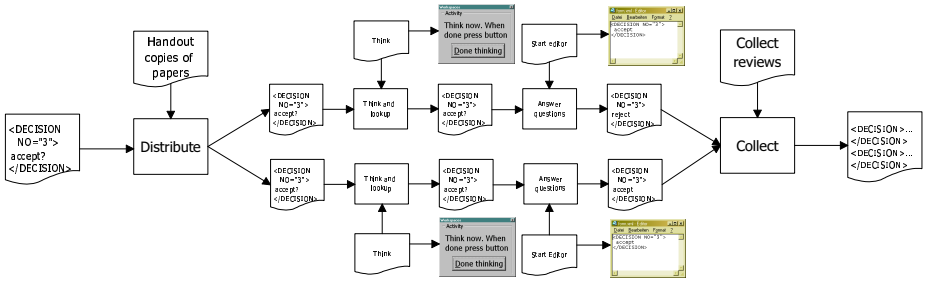


Fig. 2. A WorkSpaces workflow for reviewing conference submissions

Each step is described in another document, the step document. It contains an XSL script interpreted by the WorkSpaces engine to automatically transform documents, eg. the “Collect reviews” step, to call external applications as with the “Answer questions” step, or to wait for events external to the system as seen for the step “Think and lookup”. XSL scripts are valid XML documents.

The Workspaces engine utilizes XMLSpaces as shown in figure 3. First, some work description is retrieved with an *in* by requesting an XML document that matches the step DTD. Then, the necessary application specific XML document is requested by referring to some identifier in an attribut of a tag of the document. Then, the actual work is performed as described by the step document. Finally, the changed document is put back to the XMLSpaces with an *out* operation.

During execution of a workflow, one might try to retrieve “*something* to do”, which means a document that follows the DTD used for the description of steps. If, however, a specific task is to be done on a specific application document, one wants that *one* XML

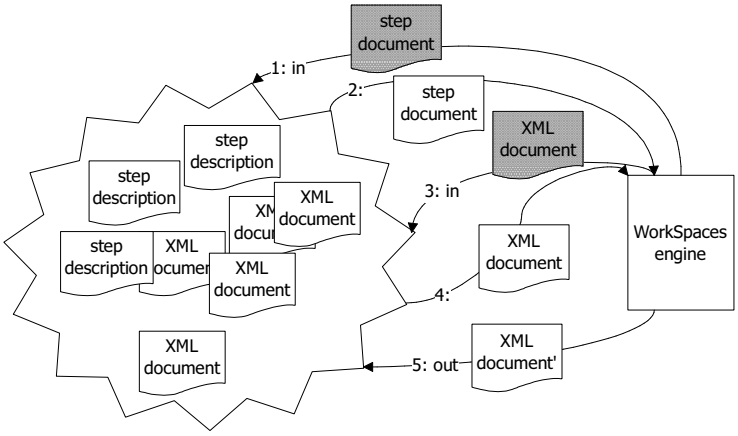


Fig. 3. Usage of XMLSpaces in Workspaces

document that might be described by an identifier in an attribute. This requirement induces the need for support of multiple relations used in matching.

The individual Workspaces engines benefit from the application of this kind of coordination technology. They are completely uncoupled and can be distributed and mobile. The number of engines participating in the system can be dynamic as new engines can join and leave at whatever time and location they want. The engine that will process future steps does not necessarily have to run when the workflow starts to execute. These attractive advantages of our architecture are due to the use of coordination technology and indicate its usefulness. The decoupled coordination style, indirected by a coordination medium that masks any issues of distribution and synchronization, thus gives huge technical freedom for a distributed and open implementation.

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. Figure 4 shows an implementation of one matching routine. Here, the XQL-engine from GMD-IPSI is used. The example shows the ease of integration of further matching routines in XMLSpaces.

While the XML standard defines one relation, namely “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.

```
package matchingrelation;
import xmlspaces.XMLMatchable;
import java.io.*;
import org.w3c.dom.Document;
import de.gmd.ipsi.xql.*;
public class XQLMatch implements XMLMatchable{
  String query;
  public XQLMatch(String xqlQuery){
    query = xqlQuery;
  }
  public boolean xmlMatch(Document xmlAsDoc){
    // forward to GMD-IPSI XQL engine
    return XQL.match(query, xmlAsDoc);
  }
}
```

Fig. 4. The implementation of XQL-matching

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

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, whether comments are 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 taken 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.

5 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 ([TR00]) 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 schemata 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 ([Fra91]).
– *Hashing*: Matching tuples and templates are stored at the same server selected by some hashing function ([Bjo92]).

XMLSpaces does not prescribe one specific approach but encapsulates the distribution strategy applied in a *distributor object*. It takes care of the registration of a server in the distributed space and offers distributed versions of the coordination primitives at its interface as shown in table 2.

Method: Explanation
register: Registers the server in the distributed tuplespace
deregister: Deregisters the server in the distributed tuplespace
distributedWrite: Performs an out of the argument tuple to the distributed space
distributedWaitToTake: Performs an in from the distributed space with the argument template
distributedWaitToRead: Performs a rd from the distributed space with the argument template
distributedEventRegister: Registers for the distributed event described in the argument
distributedEventDeRegister: Deregisters for the distributed event described in the argument

Table 2. The methods of any *Distributor* object

The implementation of the distributor object implements a distribution strategy by the respective versions of these methods. XMLSpaces servers are configured with a distribution strategy at startup. At that time, a respective *Distributor* object is created

for the XMLSpaces server, whose *register* method is used to make the server known to remote ones. Exact details of this process are specific to the chosen distribution strategy.

The registration of a server at a remote server is handled by an object of the class *RemoteTSSReception*. Depending on the distribution strategy, it registers the new server and provides it with a reference to an object of class *RemoteTSSReceiver* to which further communication on the distributed coordination primitives is directed.

Figure 5 shows the resulting configuration of objects. After exchanging the initial messages R1 and R2 with the *RemoteTSSReception*, the *Distributor* object now has references to the local space, and to remote spaces as required by the distribution schema. All coordination primitives directed to the local XMLSpaces server are redirected to the distributor object. There, the primitives are implemented by local and remote accesses as given by the strategy.

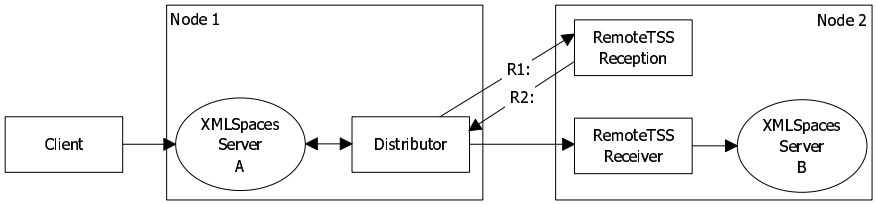


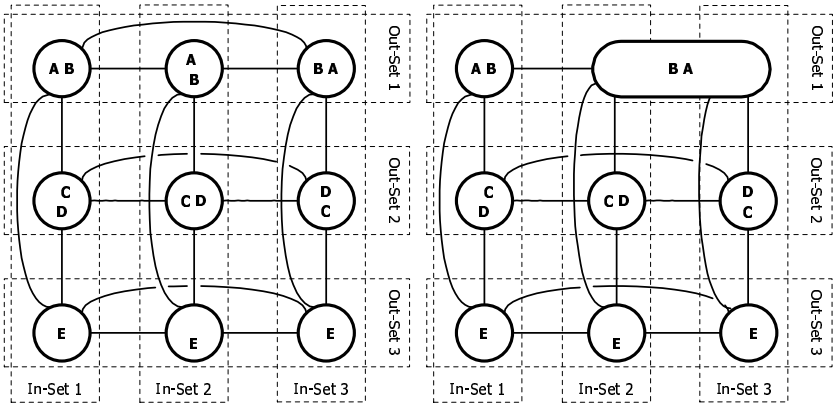
Fig. 5. Server on node 1 connecting to server on node 2

In the case of a distributed strategy without replication, for example, an *out* is implemented as a simple local *out*, while an in first searches locally and then tries to retrieve a match remotely. For a centralized scheme, the *Distributor* would simply forward the primitives to the central server. Replication schemas are implemented using methods to add, search, lock and delete tuples offered at the *RemoteTSSReceiver* object.

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 6. The nodes in the distributed XMLSpaces each store a subset of the complete contents of the data store. The nodes organized in so called *out-sets* contain identical replicas of a subset as in figure 6(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 and 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 [Tol98], XMLSpaces does not use a software-bus for communication but members of a set communicate point-to-point.



(a) The grid (b) Simulating nodes

Fig. 6. Partial Replication

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 6(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.

In XMLSpaces, there is exactly one special node, the *receptionist*, that decides one-at-a-time how new nodes join the structure. It can but does not have to be colocated with a tuplespace server. The receptionist knows about the complete structure of nodes in the system. When a new node joins, it asks the receptionist for its place in the network. Based on heuristics on the fraction of simulated nodes or considerations about the communication efficiency in the in- and out-sets, the receptionist decides on a place in the grid-structure. It informs the new node about the nodes in the target in- and out-sets and perhaps on the address of a simulated node that it shall replace.

There are three possible situations for the placement of a new node:

1. The new node is to replace a simulated node. The new node thus has to copy the state of some existing node in the out-set and stop all operations on the out-set. Then, it registers with all out-set nodes in order to participate in future operations. For registration in the in-set, all nodes there have to be informed that messages shall not be sent to the simulating node but to the new one. When this change is acknowledged, the integration of the new node is complete.
2. The new node is the first one in a new in-set. The receptionist first generates a new in-set which completely consists of simulated nodes, one for each out-set. After that, the new node replaces one of the simulated ones as described.
3. The new node is the first one in a new out-set. It then has to simulate all other nodes in that set. To do so, it registers with all in-sets to complete the integration.

When nodes are to leave the structure, there are two main variants:

1. There are other real nodes in the out-set. In this case, the receptionist decides, what other node will simulate the leaving node and the nodes it simulates in turn. As all nodes are in the same out-set, there is no state to be transferred. The leaving node has to inform the nodes in its in-set about the simulating node. After it deregistered with the nodes in its out-set, it has left the structure.
2. The leaving node is the last real node in the out-set. The leaving node then has to deregister with all in-sets it is member of as a real or simulated node. Any remaining data can be moved to some other out-set and the node has left the structure.

It turns out, that the strategies distributed and full replication as mentioned above are special cases of partial replication: The distributed strategy uses only one in-set while full replication uses only one out-set. They can be implemented by changing the decision of receptionist on the placement of new nodes in that respective one set. XMLSpaces offers respective subclasses of the *Receptionist* class. There is no need to introduce additional subclasses of *Distributor.*

6 Distributed events

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. As working with distributed events depends on the distribution strategy used, it is implemented in the *Distributor* object.

Supporting distributed events requires mechanisms to register for events, to unregister, notifying about events and handling registered events when integrating new nodes into the grid-structure.

In the case of partial replication, registering and deregistering for events has to be done on all nodes of the in-set. Events are delivered either locally to clients or forwarded to servers in the in-set, that inform their registered clients.

When a new node joins the system by replacing a simulated node, it has to copy all event registrations along with the state. If it forms a new in-set, there are no registrations that have to be considered. Finally, if it forms a new out-set, it has to synchronize with all event registrations on all of its in-sets.

When leaving a system, all locally and remotely registered events have to be deregistered. If the node was the last in its out-set, the registrations can simply be deleted, as no more events will happen. If the leaving node will be simulated afterwards, all registered events have to be transferred to the simulating node.

As with the distribution strategy, it turns out, that distributed and full replication are special cases of partial replication also with respect to events and thus, the distributor implementation can remain unchanged.

7 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 ([WMLF98]).

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 ([Wor98a]), level 1, serves as the internal representation of XML documents in actual fields. This leads 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.

8 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 ([CLZ00]) is an implementation of an extended JavaSpaces ([FHA99]) 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 closely relates tuples and Java objects and does not look at arbitrary relations amongst XML documents.

XSet ([ZJ00]) 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 [Mof99] describes a preversion for an XML-Spaces. 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 ([WMLF98]). 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.

9 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 ([Wor00a]) to represent XML documents. This standard supports XML Namespaces ([Wor00b]) 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 ([DHN00]). 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.

Efficiency and scalability of the runtime system has not yet been evaluated. Currently, the system uses RMI for the communication amongst nodes. Using a direct TCP or better UDP protocol for this purpose would speed up the communication.

http://www.cs.tu-berlin.de/~tolk/xmlspaces gives further details about XMLSpaces.

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

References

[Bjo92] Robert Bjornson. *Linda on Distributed Memory Multiprocessors*. PhD thesis, Yale University Department of Computer Science, 1992. Technical Report 931.

[CG89] Nicholas Carriero and David Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, 1989.

[CLZ00] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. XML Dataspaces for Mobile Agent Coordination. In *15th ACM Symposium on Applied Computing*, pages 181–188, 2000.

[DHN00] Stéphane Ducasse, Thomas Hofmann, and Oscar Nierstrasz. OpenSpaces: An Object-Oriented Framework For Reconfigurable Coordination Spaces. In António Porto and Gruia-Catalin Roman, editors, *Coordination Languages and Models*, LNCS 1906, pages 1–19, Limassol, Cyprus, September 2000.

[FHA99] Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces principles, patterns, and practice*. Addison-Wesley, Reading, MA, USA, 1999.

[Fra91] Craig Fraasen. Intermediate Uniformly Distributed Tuple Space on Transputer Meshes. In J.P. Banâtre and D. Le Métayer, editors, *Research Directions in High-Level Parallel Programming Languages*, number 574 in LNCS, pages 157–173. Springer, 1991.

[GC92] David Gelernter and Nicholas Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.

[Gla00] Dirk Glaubitz. Verteilte Linda-artige Koordination mit XML-Dokumenten. Master's thesis, Technische Universität Berlin, 2000. In German.

[Mof99] David Moffat. XML-Tuples and XML-Spaces, V0.7. http://uncled.oit.unc.edu/XML/XMLSpaces.html, Mar 1999.

[PA98] G. Papadopoulos and F. Arbab. Coordination models and languages. In *Advances in Computers*, volume 46: The Engineering of Large Systems. Academic Press, 1998.

[TG01] Robert Tolksdorf and Dirk Glaubitz. XMLSpaces for Coordination in Web-based Systems. In *Proceedings of the Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises WET ICE 2001*. IEEE Computer Society, Press, 2001.

[Tol98] Robert Tolksdorf. Laura - A Service-Based Coordination Language. *Science of Computer Programming, Special issue on Coordination Models, Languages, and Applications*, 1998.

[Tol00a] Robert 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.

[Tol00b] Robert 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.

[TR00] R. Tolksdorf and A. Rowstron. Evaluating Fault Tolerance Methods for Large-Scale Linda-Like Systems. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, 2000.

[TS01] Robert Tolksdorf and Marc Stauch. Using XSL to Coordinate Workflows. In U. Killat and W. Lamersdorf, editors, *Kommunikation in Verteilten Systemen (KiVS)*, Informatik Aktuell, pages 127–138. Springer Verlag, 2001.

[WMLF98] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[Wor98a] World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, 1998. http://www.w3.org/TR/REC-DOM-Level-1.

[Wor98b] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C Recommendation, 1998. http://www.w3.org/TR/REC-xml.

[Wor00a] World Wide Web Consortium. Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation, 2000. http://www.w3.org/TR/DOM-Level-2-Core.

[Wor00b] World Wide Web Consortium. Namespaces in XML. W3C Recommendation, 2000. http://www.w3.org/TR/REC-xml-names.

[ZJ00] Ben Yanbin Zhao and Anthony 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.