xSpace – A Tuple Space for XML & its application in Orchestration of Web services

Umesh Bellur  
IIT Bombay  
Powai, Mumbai 400076  
India  
+91 22 25767865  
umesh@it.iitb.ac.in

Siddharth Bondre  
IIT Bombay  
Powai, Mumbai 400076  
India  
+91 22 25764977  
siddh@it.iitb.ac.in

ABSTRACT
Today’s extended enterprise calls for the integration of several disparate systems built using multiple technologies and executing across firewall boundaries. The integration is usually done in the context of workflow orchestration and in its current incarnation, this extends across multiple enterprises over the internet. Given the disparity of these systems that need to be integrated in the workflow, these systems are usually exposed as web services to bring about a common denominator. By implication, there is a need to transfer XML documents asynchronously from one system to another since XML is the language of web service interaction. An ideal way to accomplish this asynchronous interaction is to use the concept of tuple spaces which provide a distributive, associative shared memory concept. Usually tuple spaces have dealt with the either simple types or language specific objects as tuple attributes and do not span enterprise boundaries but our need for exchanging XML documents over the internet forces us to reconsider how spaces are architected. In this paper, we present xSpace – a tuple space that deals with XML documents natively and is distributed across the internet. We also show how it can serve as a vehicle to orchestrate web services by providing an asynchronous interaction paradigm.

Categories and Subject Descriptors
D.3.3 [XML, Distributed Service oriented systems]: XML, Tuple Spaces, Asynchronous, distributed query processing.

General Terms

Keywords
XML, enterprise application integration.

1. INTRODUCTION
With the standardization of XML as the format of choice for inter

application communication, it has become necessary to investigate middleware concepts that are XML centric as opposed to say language objects such as those in Java or C++. Existing middleware such as J2EE, .NET and CORBA are focused on providing synchronous remote access to objects described using a canonical interface definition language or IDL. Even message oriented middleware such as Java Messaging Service which decouples the client and server through asynchronous interactions are based on the concepts of messages which are encapsulations of language objects that are conformant to a previously defined IDL interface. However, while most of these concepts are suitable for intra-enterprise integration, the extended enterprise today calls for the integration of systems across enterprise boundaries. B2B systems are one manifestation of such integration. Having IDL interfaces which are known apriori may be impractical when this workflow extends across the boundary of the enterprise. Instead the notion of “service orientation” has come into play where services are searched for and used dynamically with no apriori knowledge of what the interface for the service may look like. Web services are a practical manifestation of the service orientation paradigm and use XML as a vehicle for describing services, requests for services, using the services and for transmission of any parameters and results needed for and returned by the service. Web services also present an asynchronous paradigm of computation with callbacks because of the unreliability of the internet infrastructure. Most future system architectures will expose services as Web services accessible using XML and some standards similar to SOAP.

What would be useful in such an environment is middleware that can asynchronously transfer these service requests (in XML) from the user to the provider of the service. One approach is to extend messaging middleware to become XML centric. These are mainly shared queues with publish subscribe characteristics that export FIFO semantics. However it is apparent that these queuing systems do not fulfill the needs of integration where associative constraints may have to be met. We have chosen the approach of distributed associative shared memory with notification capabilities also known as tuple spaces as the alternative approach. The reasons for this change in paradigm is that tuple spaces expose both the distributed computational model supported by message oriented middleware as well as that of a shared memory computational model that drives parallel programming over a distributed environment. A space can therefore serve as a replacement for such a MOM which allows an application to place requests in the form of tuples into the space and other applications or service providers to extract these requests in any
order that they wish to look up. In a space, extraction is mainly controlled by an associative match with the structure of the tuple as well as tuple field values and can also be made to exhibit FIFO characteristics if needed.

\textbf{xSpace} is the result of applying XML centric concepts to the more generalized notion of a Tuple space as well as distributing this on internet scale by using peer to peer concepts and protocols. Originally the tuple space as conceived in Linda\cite{1} was meant to act as a distributed, associative shared memory platform on which to build applications (single node applications that took advantage of memory on other machines). Linda promulgated a generic tuple space, where tuple fields were mainly primitive values and associative lookup was done based on matching values and wildcards. JavaSpaces\cite{12} took this one step further where the tuple was a Java language object and the fields were the attributes of a object. Javaspaces was able to effectively exploit the type system of Java and allow polymorphic matching on interfaces. All of these efforts also looked only within an enterprise context and did not support internet protocols.

A tuple space for XML documents is now coming into it\'s own with a few efforts recognizing the need for XML centric middleware to replace MOMs\cite{2,7}. Most of these are hybrid efforts which allow programming language objects as well as XML documents to exist in the space but treat both as language specific objects. In this paper we lay out the architecture and discuss critical design decisions that allow optimal performance of a pure XML space that can work over the internet to span enterprise boundaries.

The rest of this paper is organized as follows. The next section talks about the related work in this area in greater detail. We then discuss the key design decision points in the construction of a XML Space, overall architecture and performance analysis. Following that we lay out a case study that illustrates the use of the XML space in business process orchestration. We conclude with status and a view to the future.

2. RELATED WORK

The concept of a tuple space was introduced originally in Linda\cite{1}. In this form, tuples consisted of simple values of the types available in and could be constructed from the programming language itself. Linda introduced the simple operations of writing a tuple into the space, looking up something in the space using a query that supported wildcard based matching and removing a tuple from the space. JavaSpaces\cite{12} technology is a simple unified mechanism for dynamic communication, coordination, and sharing of objects between Java technology-based network resources like clients and servers. In a distributed application, JavaSpaces technology acts as a virtual space between providers and requestors of network resources or objects. This allows participants in a distributed solution to exchange tasks, requests, and information in the form of Java technology-based objects. The JavaSpaces technology is written in the Java language, and is a Jini technology service. TSpaces from IBM is a set of network communication buffers called tuple spaces and a set of APIs (and classes that implement the API) for accessing those buffers\cite{13}. TSpaces allows heterogeneous, Java-enabled devices to exchange data with little programming effort. The package includes server software that implements the buffers and client software for accessing the buffers. TSpaces provides group communication services, database services, URL-based file transfer services, and event notification services. The MARS-X coordination architecture, implemented as an extension of the MARS architecture\cite{7}, defines a Linda-like middleware model to enable agent (specifically, mobile Java agents) to coordinate their activities via Linda-like access to shared spaces of XML documents. Unlike XSpace, which operates at the granularity of XML documents, MARS-X adopts a more fine-grained approach, and considers any XML document in terms of unstructured sets of tuples. XMLSpaces.NET\cite{2} implements the Linda concept as a middleware for XML documents on the .NET platform. It introduces an extended matching flexibility on nested tuples and richer data types for fields, including objects and XML documents.

2.1 Differentiation

As will become evident through the rest of this paper, there are several key distinguishing characteristics of xSpace effort that are laid out in this section.

1. xSpace is a document-centric approach and treats tuples as XML documents that are stored XML natively. Whereas TSpaces, XMLSpaces.Net, JavaSpaces are all object-centric approaches and treat tuples as language objects that embed XML at an attribute level. Storage is in the binary form of the object in these efforts. The object-centric approach has limited utility on the Internet and in heterogeneous environments since it is bound to a particular programming language and binary protocol. Whereas XSpace is meant to be used as a distributed associative shared memory for XML documents over the internet and in environments where integration of disparate applications written in multiple languages is needed.

2. The tuple is an XML document and is the unit of entry and removal in XSpace whereas other efforts permit fragmentation of the document and accept a tuple that is a mixture of objects, values and XML documents. This would mean inefficiencies in storage and query processing which we overcome by our approach.

3. We support querying against structure as well as content even though we don\’t enforce a strict type system unlike object based tuple spaces. No other existing space supports querying based on XML schema while we support both XML schema as well as DTDs.

4. We have combined the efficiencies of a XML DB gotten by indexing documents based on elements and attributes with the simplicity of a tuple space. In summary, our effort yields a structure in between a file system for XML documents and a full fledged native XML database.

3. xSpace ARCHITECTURE

This section discusses the key architecture decision points while putting together xSpaces. For the purposes of this discussion, the space is a logical entity – its physical representation is typically distributed on multiple machines or “nodes”. The space itself supports the same simple interface consisting of just 4 operations – \texttt{write} (puts a XML Document into the space), \texttt{read} (returns a document matching certain associative lookup criteria), \texttt{take} (removes the document from the space while returning it) and
The Space interface (which is a cluster by itself) is the web service handler (or SOAP server). Client requests first come to xSpace at this point. This also serves as a distribution controller – for inserts into the space, it decides to which SG the document will go to.

For querying the space, it multicasts the request to all SG by using policies on SG assignment to a node, group creation and notify the JXTA group communication protocols. Each SG replies with its local results and the space Interface is then responsible for dissolving, Load balancing and Write can be injected into space matching the criteria entered into the space).

N: Node Managers
S: Storage Nodes

Figure 1: Distribution Architecture of xSpace

xSpace is built on a peer group architecture. In fact, we are using JXTA for the purposes of the P2P infrastructure underlying xSpace. Figure 1 shows the different elements of the space. We support the concept of nodes that can join and leave a space. When the node joins a space it is assigned to a Storage Group (SG). Each node in a SG replicates documents for the sake of robustness. The specific SG to which the node goes is the decision of the node manager which is driven dynamically through administrator supplied policies (load balancing, security domains etc.). A node can belong to multiple spaces but to only one SG in a space. The space itself is federated over multiple SGs. The XML documents entering the space are distributed amongst the different SGs but replicated to all nodes within a SG. Peers (Node Managers) within Node Manager Group (NMG) work to enforce policy based decisions like nodes joining, a node leaving, SG creation, group dissolution etc. The space is programmatically managed by MBeans1 with the help of administrator supplied policies on SG assignment to a node, group creation and dissolution, Load balancing and Write can be injected into space using MBeans. Our current prototype supports round robin policy for Write, FCFS for Storage Group assignment and a MAX limit for cardinality (number of Storage Nodes) of SG.

The Space interface (which is a cluster by itself) is the web service handler (or SOAP server). Client requests first come to xSpace at this point. This also serves as a distribution controller – for inserts into the space, it decides to which SG the document will go to. For querying the space, it multicasts the request to all SG by using the JXTA group communication protocols. Each SG replies with its local results and the space Interface is then responsible for collating these results and sending it back to the client. Note that the space interface is stateless and can itself be replicated in a cluster for redundancy and load balancing.

4. Architectural HIGHLIGHTS

The key features of the XML Space as well as interesting aspects of the design are described in this section.

4.1 Space Interface

While most tuple space implementations support the critical operations of insertion, reading and deletion or taking, we also support the following:

a. Read all documents matching a set of search criteria. This returns a set of XML documents as opposed to Read which returns one document (the first in the set of matching documents).

b. Similar to a., the space supports a TakeAll operation that removes all matching entries from the space.

c. Notification: We support the notion of notification where a client can register interest in a particular kind of entry appearing in the space (subject to the same associative criteria as for querying for entries in the space) and be notified when the event does occur. The notification occurs in the form of a call back. Notifications are transactional in the sense that if the registrant is not available at the time of the event occurrence retries are provided for. The maximum number of retry attempts can be configured.

d. Multiple reads (or takes) with the same criteria in a single session will return different documents of the set rather than the same document.

4.2 xSpace as a Service

The space interface is exposed as a web service for asynchronous interaction in addition to having language specific interfaces for Java. So for an application written in Java that needs synchronous interaction with the Space, it can use the provided Java packages that contain client interfaces to the Space. Individual space operations are transactional. We also provide session based access in which a set of space operations can be tied together as a single transaction.

4.3 Distribution, Replication & Caching

xSpace is distributed with partial replication. We achieve this by the use of the notion of peer groups where a node joining the space will be assigned to a peer group depending on load and other factors. A peer group is a set of nodes belonging to the space on which all the documents assigned to the group are replicated. The index information explained later in this section is also replicated on all nodes of a peer group. Peer groups themselves form a federation and the set of all XML documents in the space are distributed amongst the peer groups. Hence we have partial replication of both the documents and the indexing information ensuring high availability. Peer groups can share nodes and do not have to be disjoint. The addition of an entry into the space is sent to one of the peer groups depending on the policy(which may be driven by factors like the number of documents and size of the documents being stored, number of notification registrations etc.). We employ a strict consistency model with the replication for now. We will build in the flexibility to replace this with another model at startup.

XML documents which form the entries of the Space are usually stored on secondary storage. However a cache exists with every

---

1 An MBean or management bean is an entity in the Java Management API which allows remote management of any entity.
peer group that is based on locality. The actual cache replacement scheme and the size of this cache is configurable.

### 4.4 The xSpace Query Language

We allow for significant degree of flexibility in terms of querying on the space. At the highest level, we the space supports querying based on either structure or on content. Structurally the query can be based on either a DTD or on a XML Schema. However we do NOT require every entry to be based on a specified DTD. We handle this flexibility by breaking down the document into a set of indices based on the structure of the document and storing only the indices and a pointer to the actual document from the index tables. This technique also allows querying based on partial structure specified as a sub-DTD.

The use of regular expressions may seem to be nothing but XPath but it must noted that, while XPath is a set of syntax rules for defining parts of an XML document, the queries over xSpace are to be evaluated over an entire set of documents. The output of a XPath query is a set of node or set of fragments within that document, but for the xSpace it is set of matching documents. So for a query like

\[ Q1: /students/student[@id=10], \]

a XPath implementation would assume that for the document in context the root element is 'students' and it has to reach its children element 'student' satisfying the predicate over attribute. But for xSpace the same query will search for all documents with root element as 'students' and with child as 'student'. The syntax of queries in xSpace is to some extent similar to XPath with following exceptions:

- a. We do not support queries which are not safe i.e., restricted in terms of scope. For example:
  \[ Q2: //text()="foobar" \]
- b. We support wildcards for values (content), but not wildcards for element or attribute names.

#### 4.4.1 Structural queries:

These queries are meant for querying the space based only on structure i.e., element/attribute relationships. For example:

\[ Q3: (/books/_*/figure | /reports/_*/figure) \]

This query will search xSpace for all documents with:

1. Elements books and figure in ancestor-descendent relationship and books being root. It could be either books\_chapter\_1/figure or books\_figures etc.
2. Elements reports and figure in ancestor-descendent relationship and reports being root.

Another example:

\[ Q4: //dealer[@ISO] \]

This query (Q4) will search xSpace for all documents having an element dealer with an attribute ISO. It could be /dealers/dealer[@ISO=9001] or /dealer[@ISO=9000]

#### 4.4.2 Content based queries:

These are meant querying based on value of attributes or elements. For example:

\[ Q5: (//[@*="sandy"] | //text()="sandy") \]

The query (Q5) will search xSpace for all documents having an element-value or attribute-value 'sandy'.

\[ Q6: (//[@*="sandy"] | //text()="sandy") | (//[@sandy]) \]

This query (Q6) will search xSpace for all documents having an element-value or attribute-value 'sandy' or element or attribute named 'sandy'.

#### 4.4.3 Hybrid queries:

\[ Q7: /students/student[@id>30]/-*/course/text()="TCP/IP" \]

\[ Q8: /dealer[@location='bandra']//product[@price<10000]/text()="herohonda" \]

The query (Q8) will search xSpace for all documents having the root element dealer with attribute 'location' as 'bandra' and with some descendent (may not be immediate child) element 'product' whose attribute 'price' is less than 10000 and its value is 'herohonda'.

If a DTD is specified along with some regular path expression then the xSpace query engine will take advantage of the fact that only those documents conforming to the schema need to be dealt with. These documents are further matched against the regular path expression and further filtered.

### 4.5 Storage and Indexing

We employ two hash tables as indices, an ELEMENT INDEX and an ATTRIBUTE INDEX. Along with this we keep a separate mapping of document identifiers to the physical location of documents which may on secondary storage or in the cache. The Element Index is hashed on Element Name and the associated entry is a list of Document IDs which contain that element. Similarly, the Attribute Index is hashed on Attribute Name. As soon as a document is written on the space, the peer group which will host this document is dynamically determined based on some property like load patterns. As a document arrives on a peer, it parses the document and adjusts all the hash entries.

### 4.6 Distributed Query Processing

Processing of the query is done in a distributed manner as follows:

- a. The query arrives at a node which runs the Space Interface as described earlier.
- b. The node multicasts the query to the other peer groups (to the primaries of the group, not the entire group).
- c. At each node to which the query has been sent, the algorithm is as follows:
- d. Parse the query, which itself is in form of XML.
- e. Go through the Element/Attribute Index for each Element/Attribute in query and return the set of documents associated with respective Element/Attribute.
f. Take the intersection of all sets considering order of the specific elements and attributes into account. This is in effect recreating the structure dynamically without the content present in the original document.

g. Run XPATH query over the set of potentially valid matches. XPATH allows us to match against partial content specified by the XPATH expression that we can construct having the partial content to match against.

h. Return set of documents which match to originator node which runs the Space Interface web service.

i. This interface consolidates results from all peer groups and hands the result back to the user.

Other design aspects include Notifications, security in terms of access control and encryption, name spaces for space decomposition and leasing where entries get invalidated after the expiry of the lease period.

5. Performance Analysis

In this section, we present the result of exhaustive performance benchmarking we have done on xSpace. The performance of xSpace depends on the following key factors:

- **Query Complexity**: Query Complexity is the number of elements and attributes in the Query. A Query with ‘k’ elements and attributes generates ‘k’ hash lookups and induces a cost of (k-1) intersections on query execution time. Worst case scenario occurs when there is no significant reduction of result set in each of the (k-1) intersections, rendering these intersections redundant. Figure 2 shows the Worst case behavior of Query Engine from the perspective of Query complexity, for a total of 2000 documents in space.

- **Number of Storage Groups**: As our performance charts illustrate, query performance scales with increase in number of Storage Groups (i.e. distribution aspect), since the documents are federated amongst the group and so is the computational load.

- **Performance for Write Operation**: Performance of write operation depends on the number of Storage Nodes in the target Storage Group (Replication cost) as well as the complexity of document (i.e. the number of elements and attributes).

![Figure 2: Performance of Query Engine](image)

**Figure 2: Performance of Query Engine**

Performance tests for xSpaces were conducted on four 2.8 GHz Pentium-IV Machines with 512 MB RAM running Linux platform. Following test schemes were used to analyze the Worst case performance of core operations (i.e. read/readAll, take/takeAll, write):

- **Test Scheme for Write**: In order to measure the scalability and performance in write operation, space was load tested with 100 concurrent write operations, with varying complexity of documents. Result are shown in Figure 3.

![Performance Chart for Write Operation](image)

**Figure 3: Performance of Write Operation**

- **Test Scheme for Read/Take**: Several experiments were conducted to evaluate the performance of read/take operation. Space was initialized with 10000 documents and time was measured for varying number potential matches. Two documents are potential matches if and only if, both of them are a valid result for a particular query. Results are shown in Figure 4 and 5. Note that the time to get to the first document dominates because the set intersection operations for narrowing the potential matches. The query complexity in our case is maximum which accounts for the high response times even to return the first document. If we drop the query complexity, we have a linear drop in response time as shown earlier.

![Performance Chart for Read Operation](image)

**Figure 4: Performance of Read Operation**

Performance of ReadAll and TakeAll is affected by the fact that space has to wait for evaluation of query on complete set of
documents, which is not the case with read where space returns immediately with the first document found.

Our analysis shows that xSpace performs 3-4 times better than the existing systems like XMLSpaces.NET [2] for similar sizes of data sets even though we have distribution over a wide area network. The focus of our current work is to reduce the cost of intersection, which will bring down the query execution time significantly (order of magnitude) in worst case scenarios. Our best case scenarios are in less than 10s of milliseconds which is entirely acceptable.

6. xSpace as an Associative Memory Grid

Just as Grid Computing seeks to pool computational resources to accomplish complex computations, we can envisage the notion of a memory grid that will pool distributed memory into a cohesive whole and then lend associative semantics to it. There can be applications that are not computationally intensive but need to handle large amounts of data in memory. As long as the data access times over the network are faster than going to stable storage (this will be the case when the data structures have to be allocated and initialized dynamically adding to the latency of retrieving data from stable storage), this paradigm of a memory grid can serve as a powerful alternative to stable storage where these access times are critical. We are currently prototyping this notion and seeking applications that can use this concept.

7. Applying xSpace to orchestrating business process executions

We have applied the xSpace for different kinds of applications – one being business process orchestration and the other being a memory grid. In this section, we lay out the case studies in detail and show how an XML Space would be useful in solving the problem.

7.1 Orchestrating Workflow with XSpace

Consider a case of a mid-size make-to-assemble manufacturing firm that follows a business process for servicing its customers. Here the completion off each step triggers the next step. For simplicity, we have omitted exception paths of the workflow: This above business process is usually translated to an executable workflow by defining control logic that takes the order through the right path (there’s only one path in this process) as well as mapping individual tasks to various systems that are able to perform the tasks. This is shown below:

1. When the customer places the order, the Sales Department records the order details using an Order Management System.
2. The Production Department uses an Inventory Management System to check the inventory and to issue the required material. It places orders for unavailable components, if required.
3. The Production Department initiates the manufacturing of sub-assembly and the final assembly by using a Production Planning & Control System
4. When the customer places the order, the Sales Department records the order details using an Order Management System.
5. The Production Department uses an **Inventory Management System** to check the inventory and to issue the required material. It places orders for unavailable components, if required.

6. The Production Department initiates the manufacturing of sub-assembly and the final assembly by using a **Production Planning & Control System**.

7. The Dispatch Department delivers the finished goods to the customer and uses **Billing System** to raise the invoice.

8. The Accounting Department records various transactions happening throughout the execution of this business process by using the **Accounting System**.

Each of these systems is exposed as a web. As it can be seen, all the systems involved need to interact with each other for executing this business process. These interactions can be hard wired which will embed the business process, the preferred design accomplished using the Space. xSpace is employed here for xSpace to put in requests for the next workflow task and reads the results. Different kinds of XML documents describe the results of different steps and the controller does an associative search to determine whether any of the known document types are available for processing. Notification adds efficiency over polling here since the controller can be notified about the availability of any result. In fact, this notion allows us to distribute the controller functionality since different parts of a federated controller can be responsible for different parts of a workflow.

**8. STATUS & FUTURE WORK**

Currently work is going on to build the notification system before releasing it into Open Source community. We expect to complete the notification system by the time this goes to publication. The two case studies outlined in this paper serve as examples which we will distribute along with the Space code. Future work consists of adding namespaces and security to this software and investigating the resulting effect on performance. We are also looking into automating business process orchestration using the space as a communications bus.

**9. CONCLUSIONS**

Today’s model of coordination using message queues is under threat because of the changing nature of the systems. The need to coordinate business processes over internet using standards as well as have asynchronous interactions amongst hundreds of diverse applications calls for a revival of the tuple space notion. However it now needs to be considered in lights of today’s protocols for integration which include web services and XML. We have therefore focused on the efficiencies that can be wrung out of a pure XML tuple space rather than hybrid spaces that have existed earlier. The benchmark is also a contribution of this project to the open source community that can be applied to evaluate all tuple spaces for different types of queries and contents.

**10. ACKNOWLEDGMENTS**

This project has been made possible through a grant given by **L&T Infotech** for research into enterprise integration middleware which is supporting the xSpace project. We acknowledge contributions made by Sandesh Tiwari, Deepti Agarwal and Sandeep Satpal, Shruti Mahambre, Sridivei Raman and Niranjan Srinivasan and Shrikanth Mulik in designing and building the XML space. We would like to acknowledge the feedback on all aspects of this project from Sateesh Awade from L&T Infotech.

**11. REFERENCES**


