Proceedings of the 10th IASTED Int. Conf. on Software Eng. and Appl. (SEA), pp.146-152, Nov. 13-15, 2006, Dallas, USA

THE XD MODEL: EXTENDING XML AND DOM TO STANDARDS BASED COORDINATION[*]

Gerson Joskowicz, eva Kühn, Martin Murth
Space Based Computing Group, Institute of Computer Languages, TU Vienna
Argentinierstraße 8, 1040 Vienna, Austria
{josko, eva, mm}@complang.tuwien.ac.at

ABSTRACT
Distributed IT architectures are becoming increasingly complex and hard to maintain as they evolve into sophisticated fabrics of interconnected and dependent services. The proposed coordination model XD describes the foundation for a coordinated XML based communication space providing a simple and robust communication paradigm. It ties into current distributed applications by offering two essential features: First, the XD coordination model does not prescribe a specific coordination model and adapts to most interaction patterns. Second, the interface employs accepted standards simplifying the adaptation of the model to current programming styles, architectures, and tools.

KEYWORDS
Coordination model, collaboration, shared data, distributed systems, XML, document object model.

1. Motivation

Until recently, middleware systems that constitute the backbone of distributed systems, dealt mainly with communication issues. These systems provide an additional layer of abstraction for remote method or service invocation, automate data serialization, and mask network faults and latency. However, during the last years, the role of coordination between network participants has been gaining in importance. Business partners not only need to exchange purchase orders and article lists, they also need to dynamically negotiate prices, to collaboratively process orders and compile offers; more generally spoken: they need to collaboratively work on data.
Current software development frameworks provide various data structures that support coordination in single-site applications. The programmer decides whether a single shared variable, a dictionary, a queue, a tree, or a higher level data structure, being a composition of lower level ones, is most suitable for solving a coordination problem. On the opposite side, the current state of the art in the design of distributed systems is based on message exchanges in a FIFO fashion, e.g. by employing message queuing systems. However, considering message queues as the only means to coordinate distributed systems constrains a system's architectural design. Service Oriented Architecture (SOA), as the most important representative of message based architectures, proposes special extensions to support coordination [1], but since coordination is about *sharing state,* SOA as a stateless communication paradigm requires additional development effort on the application layer in order to support coordination. Examples are correlation of request and response messages, explicit replication to achieve consistency of distributed data, etc.
Stateful communication, i.e. communication via shared data [2] instead of explicit message exchange, enables the implementation of the same coordination data structures as used in single-site applications and passes the choice of the most suitable coordination data structure back to the developer. We therefore propose an open, standards based, and generic coordination model that leverages the advantages of shared data spaces [3] (e.g. Linda [4], JavaSpaces [5], Orca [6], Corso [7], XVSM [8]).
The three main pillars of the XD (e**X**tensible & **D**OM based) coordination model are:

- Use of XML as a presentation layer
- Use of the Document Object Model (DOM) [9] for data access
- Integration of DOM with a suitable coordination model

XML was chosen as data presentation layer since it is a de facto standard in a wide range of application areas and the primary data format for business-to-business and application integration systems. DOM, being a well-known and standardized interface technology for XML, is thus used to access the XML data.
The remainder of this paper is structured as follows. The rest of Section 1 presents a requirements analysis for an XML-based coordination model based on a concrete use case scenario. Section 2 introduces DOM and defines the necessary extensions to make it a fully featured coordination model. Synergy effects of the proposed model and XML are described in Section 3. Section 4 gives an evaluation of the XD model. Related work is described in Section 5, and Section 6 presents future research plans and concludes the paper.

1.1 Use Case Scenario: An Auction System

The following example of an XML document (Listing 1) illustrates the idea. The example describes the data representing the current state of an auction system: a list of

[*] This work has been supported by the TRIPCOM (IST-4-027324) Specific Targeted Research Project.

articles which either belong to the group of active auctions or to the group of already sold auction articles.
By employing an XML centric coordination middleware, one could implement a truly distributed auction system. Distributed and autonomous clients would monitor the current state of the auction represented by the XML document, and would react by raising a bid or by dropping out when they observe changes in bids. Their reaction would update the state of the auction.

```
<Auctions>
   <Active>
      <Article>
         <Description> Pope's Car </Description>
         <CurrentBid> 140000 </CurrentBid>
         <Bidder> Lucy E. </Bidder>
         <ExpirationDate> 2005-10-10 </ExpirationDate>
      </Article>
      <Article>
         <Description> XBOX </Description>
         <CurrentBid> 140 </CurrentBid>
         <Bidder> Linus </Bidder>
         <ExpirationDate> 2005-10-09 </ExpirationDate>
      </Article>
   </Active>
   <Sold>
      <Article>
         <Description> Britney's Trousers </Description>
         <CurrentBid> 3200 </CurrentBid>
         <Bidder> Tommy Lee </Bidder>
         <ExpirationDate> 2005-09-01 </ExpirationDate>
      </Article>
   </Sold>
</Auctions>
```

Listing 1: XML Example for an Auction System

Based on a requirements analysis of this and other, more comprehensive use cases, the following set of quality criteria for coordination systems were defined for the evaluation of the proposed model:

- **Support for flexible coordination data structures**: Clients must not be tied to a predefined coordination pattern. E.g. bidders want to be notified about price changes of articles (publish/subscribe) as well as to raise bids themselves by updating auction articles (directly in the shared document).
- **Performance and scalability**: Enterprise level applications need to exhibit low response times and provide high scalability. E.g. neither peak loads nor an increasing number of clients should compromise the auction system.
- **Adaptability and extensibility**: The system must easily adapt to continuously evolving requirements and changing IT infrastructures. E.g. the auction system may be extended by security and privacy policies.
- **Openness and interoperability**: The coordination model should very well interoperate with other standards. E.g. integration with Web service infrastructures, third party XA transaction managers [10], etc.
- **Reliability and availability**: Network and system interruptions must not result in inconsistent data. E.g. abrupt disconnection of a mobile auction client should not hinder proper functioning of the auction system; the mobile client should be able to recover and proceed seamlessly.

2. The XD Coordination Model

The basic idea behind the XD coordination model is to reuse the DOM standard and extend it in such a way that it supports collaborative work on XML documents. Therefore, we define two main extensions: First, we enhance the semantics of the DOM event model so that it allows for the implementation of arbitrary coordination patterns. And second, we introduce a transaction model for DOM that manages concurrent access to XML data.

2.1 The Document Object Model

The Document Object Model (DOM) [9] is a specification of how an HTML or XML document is represented in an object-oriented fashion. It provides an application programming interface (API) to access and modify the content and structure of a document based on a tree-like node data structure. The API specification is programming language and platform independent.
Since version 2.0, DOM defines also an event model. Amongst other event modules necessary for reacting to key events, mouse events, page reloads, etc. in web browsers, it defines an event module called *Mutation-Event* which allows a listener to observe and to react to any mutation in an XML tree. Listeners can register to certain nodes of the tree and are notified about tree restructuring and data modifications on the node itself and on any of its sub-nodes.
This event model allows for coordination according to the *publish/subscribe* pattern. The standard specification precisely defines when an event has to be fired and when and in which order the listeners have to be notified about the event, but does not define an event distribution process for multi-threaded or distributed applications. Nonetheless, the DOM events standard can serve as a foundation for the XD coordination model. For updates to the XML tree, the standard already defines a series of event types suitable for collaboration activities:

- DOMSubtreeModified[2]
- DOMNodeInserted
- DOMNodeRemoved
- DOMNodeInsertedIntoDocument
- DOMNodeRemovedFromDocument
- DOMAttrModified
- DOMCharacterDataModified
- DOMElementNameChanged

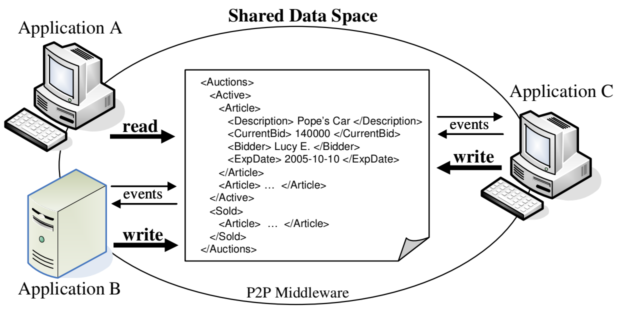
All events can be received by listeners registered at the event target node or at any of its ancestor nodes. The event distribution process consists of three different phases [9]. In the *capture phase*, the first phase, listeners of ancestor nodes of the event target can 'capture' the event, i.e. they can pre-process it before the actual event target is notified. Capture operates from the top of the tree and can be configured at the event listener. In the second

[2] This is a general event for notification of all changes to the document. It can be used instead of the other more specific events.

phase, event listeners of the actual event target are notified about the event. In the third phase, called *bubbling phase*, events with a bubble flag set are automatically propagated upwards in the tree making it the opposite of event capture. As opposed to event capturing, bubbling is a property of the event type and cannot be controlled by the listener.

2.2 Extension of the DOM Event Model

Figure 1 shows an example scenario with three applications (clients) accessing Section 1.1's auction system's XML document concurrently, two of them (applications A and C) also listening to particular events. Parallel data access necessitates synchronization techniques, which should be realized by the middleware layer. The middleware has to dispatch the events in the correct order, it has to offer a reliable event notification mechanism, and it must provide the means to put the whole communication process in a transactional context.



```
<Auctions>
  <Active>
    <Article>
      <Description> Pope's Car </Description>
      <CurrentBid> 140000 </CurrentBid>
      <Bidder> Lucy E. </Bidder>
      <ExpDate> 2005-10-10 </ExpDate>
    </Article>
    <Article> ...  </Article>
  </Active>
  <Sold>
    <Article>  ...  </Article>
  </Sold>
</Auctions>
```

Figure 1: Concurrent Data Access

2.2.1 Event Types

While most coordination models follow either a data-driven or a control-driven approach [11], XD seeks to integrate both approaches.

The DOM event model already provides the basic means for data-driven coordination but lacks some important coordination features. One of these shortcomings is that it is not possible to be notified about read access to specific data. Thus, two supplementary event types are defined in XD:

- DOMAttrRead
- DOMCharacterDataRead

These event types enable the implementation of special interaction patterns usable for *access control, monitoring, auditing*, etc. and can be employed for tracking and analysing the data flow within a distributed application.

For the integration of control-driven coordination, we also define an additional event type:

- DOMConsumableChildNodeAvailable

Use of this third new event type enables mapping of numerous coordination patterns typically applied in message queuing systems [12] (e.g. load distribution) to the XD coordination model. DOMConsumableChildNodeAvailable allows a listener to be notified about a node having 'consumable' child-nodes which is similar to the DOMChildNodeInserted event but may be dispatched at any time a node has a non-empty set of child-nodes (i.e. also without preceding insertion). The XD model does not prescribe when and which listeners have to be notified about this event, it is rather up to the middleware that implements the model to determine the best dispatching strategy.

2.2.2 Event Listener Types

In DOM, the interplay of capture and bubble defines the order of event delivery. However, this model does not suffice to determine unambiguous event ordering in distributed environments, since clients might dynamically register to the same events on the same nodes, not knowing of each other. In XD, such listeners are notified in parallel, and a transaction model is employed to handle concurrency (Section 2.3). In addition, XD provides a mechanism which permits a client to influence event distribution by introducing two additional properties for listeners. First, it allows defining whether a listener should be notified before or after an event actually takes place, and second, it distinguishes between synchronous and asynchronous listener notification. Their combination leads to the following three applicable listener types:

- PRE^{sync} : Synchronous pre event listeners
- $\text{POST}^{\text{sync}}$: Synchronous post event listeners
- $\text{POST}^{\text{async}}$: Asynchronous post event listeners

All synchronous listeners reliably receive event notifications before/after the execution of the operation that triggered the event notification process according to the sequential capture/bubble ordering (only those listeners registered to the same node are notified in parallel). They contribute to the same transaction and must complete successfully before the transaction can be committed.

Pre event listeners can perform preparation work in order to make the data space ready for the upcoming operation or to 'pre-act' to a certain kind of operation. E.g. a *data-on-demand* mechanism can be implemented with this type of event listener by updating a certain node's value prior to an explicit read request.

Post event listeners allow a process to react to data access or to monitor the effects of certain events. Asynchronous post event listeners always reliably receive notifications *after* the event processing has been successfully committed. They are notified in parallel, the control flow remains with the requestor process, and the next operation can be performed without having to wait for other listeners to complete. On the one hand, this improves the performance. On the other hand, the listeners might observe the tree in an already advanced state as subsequent transactions could already have been completed before the event was delivered to the remote listener.

A notification buffer is needed to temporarily store the occurred notifications which can only be delivered to asynchronous event listeners after successful transaction commitment.

There is no asynchronous pre event listener type available in XD. Since we allow asynchronous listeners only to report about already committed operations, this listener type would have the same semantics as the asynchronous post event listener.
The entire process of event dispatching is summarized in Figure 2. Note that the listeners in the standard DOM event model correspond to POST^{sync} listeners.

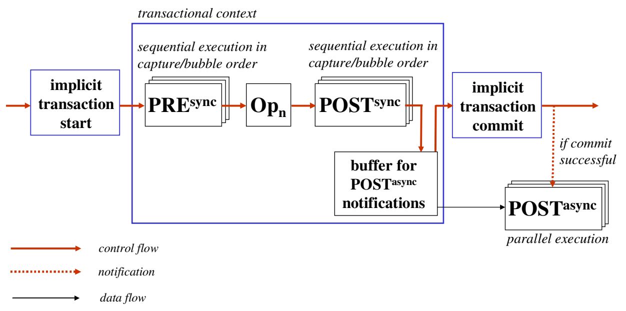

transactional context
sequential execution in capture/bubble order
sequential execution in capture/bubble order
implicit transaction start
PRE^sync
Op_n
POST^sync
implicit transaction commit
if commit successful
buffer for POST^async notifications
POST^async
parallel execution
control flow
notification
data flow

Figure 2: Parallel Listener Notification

The operation of interest is Op_n (e.g. an auction agent's raise of a bid). Upon submission of operation Op_n, all registered PRE^{sync} listeners are notified about the event. Hereby, the capture/bubble order is kept and listeners registered at the same node are notified in parallel. An implicit transaction context is instantiated and passed to the listeners. The listeners can now use this transaction context to contribute to the current operation (reflecting the still uncommitted effects of PRE^{sync} listeners that have been executed before) or they can use their own transaction context for operating on the last globally visible version of the XML tree.
As soon as the last PRE^{sync} listener returns, operation Op_n will be executed. The effects of all PRE^{sync} listeners are clearly visible to Op_n which itself will be visible to subsequently executed POST^{sync} listeners. When the last POST^{sync} listener returns successfully, the transaction will be committed automatically and the POST^{async} listeners are notified. The POST^{async} listeners can be notified in parallel as they do not adhere to the capture/bubble ordering. Subsequent operations $Op_{i>n}$ including their listeners will see the data modifications performed by Op_n and by all its synchronous listeners but not necessarily by those of the asynchronous ones.

2.3 A Transaction Model for DOM

The previous section already showed that providing a transaction model is essential for XD's event model. Unfortunately, the topic of transactions is not addressed in the DOM specification.
In XD, the function of transactions is twofold. First, they guarantee atomic, consistent, isolated and durable execution of a series of operations on the XML data, and second, sub-transactions are used to control data visibility for listeners that receive events during the capture or bubble phase. Data updates performed by synchronous listeners are visible to the enclosing top transaction and to further sub-transactions of this top-transaction, i.e. to synchronous listeners executed in a later stage of the capture/bubble phase, but not to other processes until the transaction has committed. Synchronous listeners can contribute to a transaction, they can cause its failure, but they cannot commit a transaction. Transaction commitment requires *all* the participating listener processes to have completed successfully.
It should be mentioned that synchronous listeners that employ their own transaction rather than contributing to the passed transaction context may produce side effects which would not be rolled back in case of transaction abortion. For proper handling of such cases, advanced concepts for semantic compensation [10] are needed.

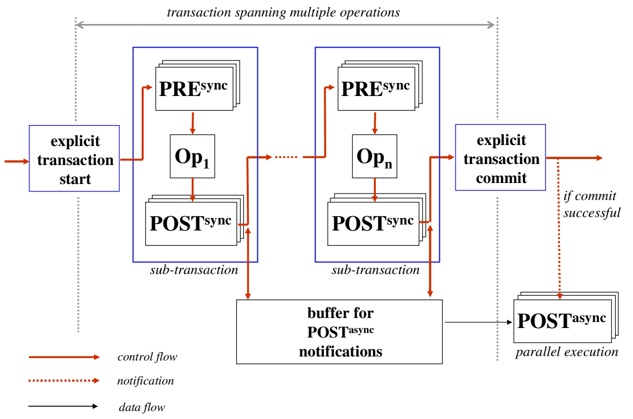


Figure 3: The XD Transaction Model

Another important issue is delivery of event notifications to asynchronous event listeners. The notification of this listener type cannot be integrated with the current transaction context directly, as the notifications can only be sent *after* an operation was committed. Thus, event notifications for asynchronous listeners are delivered as soon as the enclosing transaction has finished. Figure 3 shows the process of event dispatching for a group of operations belonging to the same transaction.

3. Synergy Effects of XD and XML

By building on XML, XD may directly adopt any standardized XML derivate. Hence, there is a great potential for extensions to this model which can be regarded as the main added-value of XD.

XML Schema for Guaranteeing Valid XML. A frequently encountered problem in entirely dynamic coordination systems is the lack of mechanisms that guarantee consistent data structures. In Linda, for example, any client is allowed to write arbitrarily structured tuples to the tuple space which may result in an unmanageable data set. For XML, there are already two standards available that allow predefining a structure for XML data, namely Document Type Definitions (DTD) and XML Schema [13]. Since the second is more expressive and employs XML syntax, we plan to integrate XML Schema validation into XD.
The validation function will also be integrated with the XD transaction model preventing the system of committing data that would violate the schema.

XML Schema for Initial Document Instantiation. XD also eases the development of distributed peer-to-peer applications communicating via a shared data space. A typical problem in such applications is that they require a preliminarily initialized data structure. In the distributed auction system, this corresponds to the existence of the elements *Auctions*, *Active*, and *Sold*. Consequently, it does not suffice to know how to use the common data structure; every peer also has to know how to initialize it. By providing an XML Schema, such initial data structures could be created automatically by employing algorithms that generate a minimal valid XML data structure from a given XSD file.

Plain XML for Data Import and Export. Plain XML can be used for the purpose of data migration, data import and export. An entire XML file or any sub-tree of an XML data structure can be inserted at any arbitrary node of a shared document. Data can be exported in order to further process the XML content with other tools in the same way. The import function could also be used for initializing a shared data space (see above).

XPath for Document Navigation. XPath [14] is a language designed for navigating an XML document. It supports finding specific nodes via user-defined predicates and is used as a query language. In the XD data space, XPath can be used to directly access individual nodes without having to manually navigate through the entire XML tree. The path expression "/Auctions/Active/Article[position()=1]", for instance, would return the "Pope's Car" *Article* node of the auction system of Section 1.1.

There are also numerous other XML standards providing useful functionality, e.g. XSLT [15] for tree restructuring or Xquery [16] as XML data manipulation and query language.

4. Discussion and Evaluation

In the following section, we discuss the proposed XD model and its prototypical implementation according to the quality criteria defined in Section 1.1.
The prototype is based on XVSM (eXtensible Virtual Shared Memory) [8], a space based coordination middleware [2] comparable to distributed file systems like Plan 9 [17] and NFS [18]. On each site of the network a part of the global data space can be located as a local data space. However, XVSM differs from distributed file systems in that it creates an abstraction over all distributed nodes by providing a uniform and simple common data space interface similar to Linda's basic operations *read*, *take*, *write* and *notify* [4]. All sites together form the entirety of the distributed coordination space which can be accessed in a transaction-safe way.

4.1 Support for Flexible Coordination Data Structures

A single XD node can serve as a generic coordination data structure that may be used as shared variable, stack, message queue etc. and composition of these simple data structures allows for building higher-level coordination data structures that can be used for the coordination of entire distributed systems. By incorporating stateful communication, all operations can also be grouped and executed in a transactional context which is typically not possible in message passing systems. However, the model can easily be integrated with message passing systems, as a queue is only one out of many available coordination data structures in XD.
Collaboratively manipulating XML sub-trees and reacting to events depicts typical data-driven coordination. For the integration of control-driven coordination, specific services can be bound to a *request/response* coordination data structure (e.g. a request node and a response sub node).

4.2 Performance and Scalability

First throughput measurements of performed single-site operations with the implemented Java 1.5 prototype running on a commodity Windows XP PC are summarized in Table 1. The results illustrate very well that the computational overhead for transactional operations significantly decreases the overall throughput rates but still allows for efficient data access. Since listener notifications involve multiple transactional read and write operations, the throughput numbers decrease accordingly.

Benchmarked Operations Pentium 4, 1.87 GHz, 1024 MB Ram	**Ops / sec**
Read Operations	9733
Transactional Read Operations	3428
Transactional Write Operations	2638
Synchronous Listener Notifications	984
Asynchronous Listener Notifications	1378

Table 1: Throughput Benchmarks

For evaluating the replication overhead, a load distribution coordination data structure was implemented. One producer process generates a data item every 10ms and appends it to the list of child-nodes of a certain XML node. Worker clients registered to this node listen to DOMConsumableChildNodeAvailable events. We let a worker client require 100ms to handle one data item; thus, multiple workers are needed to process the data items. By incrementally increasing the number of parallel worker systems, we reduce the time to process the data stream. Figure 4 shows the optimal time frame as well as the time required by the XD based implementation to process a stream of 1000 data items. Although the system scales quite well, the plot shows that the processing overhead slightly increases with the number of workers due to the additional replication effort.
Note that compared to load balancing with traditional message oriented middleware, worker clients do not have to know the recipient of the processed data. In XD, they just put the processed data item back to the shared document, which allows for the implementation of more autonomous and loosely coupled systems.

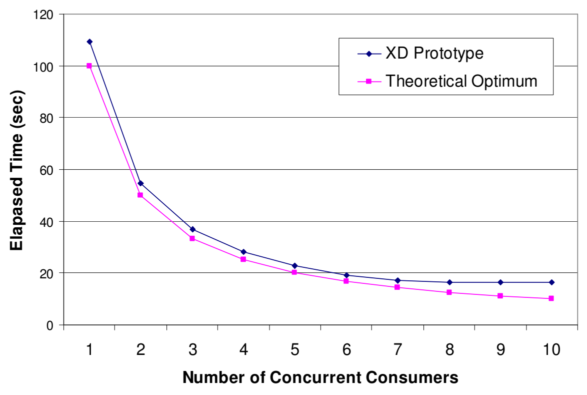


Figure 4: Replication Overhead

Since the current implementation is still considered a prototype, more comprehensive performance measurements will be performed in future work. We expect that the use of pre event listeners and read events for implementing a *data-on-demand* pattern will allow for further considerable performance improvements.

4.3 Adaptability and Extensibility

In the first place, XD implicitly inherits a certain degree of extensibility simply by leveraging XML and DOM. The shared XML data can be dynamically restructured which allows for performing updates to an active application. Upgrades or schema modifications can be simplified by using XSLT for transforming data in the XML space.
Furthermore, system components can be seamlessly added to and removed from a running system. Deployment of new components only requires registration of the components' listeners at the concerned nodes. As the supported event and listener types can be used to pre-process any kind of data access, existing infrastructure like e.g. authorization services and monitoring systems can also be integrated easily.

4.4 Openness and Interoperability

Both XML and DOM are open standards. Today, almost any available system has at least some kind of XML compliant interface which could be used to connect it to an XD implementation.
By providing the possibility to map various coordination data structures to XD, the model also eases integration of systems with very different requirements for coordination. The XD prototype provides an extended but fully compliant interface implementation of the DOM API specification.

4.5 Reliability and Availability

A crucial advantage of employing stateful communication is that it implicitly supports system recoverability. As the XML document represents the common state of the entire distributed system, a restarted client system can simply proceed on the current XML data without the risk of having lost important information.
In XVSM, a persistency mechanism guarantees full recoverability of the shared data. Moreover, XVSM makes use of a replication protocols based on end-to-end argument [19] which keep the distributed data consistent and employ advanced caching mechanisms [20] in order to mask network interruptions.
Finally, automated XML Schema validation can be incorporated to prevent accidental or malicious corruption of XML coordination data structures.

5. Related Work

XML database systems like e.g. Xindice [21] and XML:DB [22] provide a basic form of coordination middleware. However, these systems do not provide an event notification model flexible enough to support high-level coordination patterns. Moreover, they often make use of proprietary data update languages which makes integration with other systems a complex and error-prone task.
In the Linda coordination language, data is represented as a set of structured tuples. XMLSpaces [23] is an extension to this model and serves as a middleware for XML documents. Tuples can hold entire XML documents as well as single values and support more advanced matching mechanisms than basic Linda (e.g. conjunct matching with DTD, XPath and XQL [24] expressions). A drawback of XMLSpaces is that agents can only exchange entire XML documents rather than sharing XML content.
Another approach to XML-centric coordination is provided by the XMARS coordination model [25][26]. It also builds upon the Linda coordination model, but in contrast to XMLSpaces, it introduces finer granularity as it considers any XML document as sets of tuples and implements a JavaSpaces-like interface. XMARS also provides a transaction model, but the transactions are not fully integrated with the model, since they do not span the whole process of requesting a tuple and invoking a reaction method (i.e. the notification of a listener).
A quite similar approach is realized by XMiddle [27], a coordination middleware for agents communicating via shared XML data spaces. It puts its main focus on mobile application scenarios and implements sophisticated reconciliation strategies for disconnected and reconnected clients. However, XMiddle only provides a very basic event notification mechanism and lacks a common transaction support.

All the described approaches have in common that they do not provide a fully standards compliant interface and therefore cannot adopt other XML related technologies and standards as easily as XD.

6. Future Work and Conclusion

6.1 Future Work

Further research work will focus on the development of automated mappings of XML data to programming language specific data types and on the formalization of the proposed event and transaction model.
Open issues yet to be addressed are the conceptualization of an appropriate security model for XD, how to link dif-

ferent XML data spaces, and how to implement these interconnections.
In a next step, we will integrate the XD prototype with an XPath engine and perform further performance tests.

6.2 Conclusion

The proposed XD model provides an XML standards based approach to coordination. We showed that by introducing an extended event model and transactional semantics to DOM, we obtain a flexible and versatile coordination model. Composition of stateful generic coordination data types allows for the implementation of highly customized coordination data structures for distributed systems. Thus, a system's architectural design is no longer predetermined by the coordination model of the middleware layer, but can rather be tailored to the specific requirements.
The use of XML as a presentation layer makes various existing XML derivates potential candidates for useful extensions to the model while at the same time preserving standard conformity.
The prototypical implementation of a distributed XD coordination system can be regarded as a first proof of concept and will serve as a basis for further developments.

Acknowledgements

Special thanks are due to Johannes Riemer for his contributions to the development of the technical concepts.

7. References

[1] Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard, D. and Shewchuk, J., and Storey, T. Web services coordination (WS-Coordination), *Protocol Specification*, 2002.

[2] Kühn, e. *Virtual shared memory for distributed architecture*. Nova Science Publishers, 2001.

[3] Gelernter, D. and Carriero, N. Coordination languages and their significance. *Communications ACM 35*, 97-107, 1992.

[4] Gelernter, D. Generative communication in Linda, In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 7, No. 1, p.80-112, 1985.

[5] Sun Microsystems. JavaspacesTM service specification. Technical report, 2003. Available from: http://java.sun.com/products/jini/2.0.2/doc/specs/html/js-spec.html

[6] Bal, H. *The Shared Data-Object Model as a Paradigm for Programming Distributed Systems*. PhD. Thesis, Vrije Universiteit, Amsterdam, 1989.

[7] Kühn, e. Fault-tolerance for communicating multidatabase transactions. In *Proceedings of the 27th Hawaii International Conference on System Sciences (HICSS)*, ACM, IEEE, Vol. 4., No. 7, Wailea, Maui, Hawaii, 1994.

[8] Kühn, e., Riemer, J., and Joskowicz, G. *XVSM (eXtensible Virtual Shared Memory) architecure and application.* Technical Report E-185, Institute of Computer Languages, Vienna University of Technology, Vienna, Austria, 2005.

[9] World Wide Web Consortium, *Document object model level 2 specification*. Iuniverse Inc., 2000.

[10] Elmagarmid, A.K. *Database transaction models for advanced applications.* Morgan Kaufmann Publishers, 1991.

[11] Papadopoulos, G. A. and Arbab, F. Coordination models and languages. *Advances in Computers*, 46, 1998.

[12] Hohpe, G. and Woolf D. *Enterprise integration patterns: designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2003.

[13] Valentine, C., Dykes, L., and Tittel, E. *XML schemas*. Sybex Inc., U.S, 2002.

[14] Wilde, E. and Lowe, D. *XPath, XLink, XPointer, and XML.* Addison Wesley, 2002.

[15] Tidwell, D. *XSLT*, O'Reilly, 2001.

[16] Brundage, M. *Xquery*. Addison-Wesley Professional, 2004.

[17] Pike, R., Presotto, D., Dorward, S., Flandrena, B., Thompson, K., Trickey, H., and Winterbottom P. Plan 9 from Bell Labs. *Computing Systems*, Vol. 8, No. 3, pp. 221-254, 1995.

[18] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. Network file system (NFS) version 4 protocol. Technical Report, RFC 3530, IETF, April 2003.

[19] Saltzer, J.H., Reed, D.P. and Clark, D.D. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, Vol.2, No.4, pp.277-288, 1984.

[20] Kühn, e., Beinhart, M., and Murth, M. Improving data quality of mobile Internet applications with an extensible virtual shared memory approach. In *Proceeding of Iadis WWW/Internet 2005 Conference*, Lisbon, Portugal, Vol. 1, pp. 443-450, 2005.

[21] Leung, T. *Professional XML development with Apache tools*: *Xerces, Xalan, FOP, Cocoon, Axis, Xindice*. John Wiley & Sons, 2003.

[22] XML:DB, *Initiative for XML databases*. Available from: http://xmldb-org.sourceforge.net/index.html

[23] Tolksdorf, R. and Glaubitz, D. XMLSpaces for coordination in Web-based systems. In *Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, p.322-327, June 20-22, 2001.

[24] Robie, J. *XQL (XML query language). Technical Report*, 1999. Available from: http://www.ibiblio.org/xql/xql-proposal.html

[25] Cabri, G., Leonardi, L., and Zambonelli, F. XML data spaces for mobile agent coordination. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, Vol. 1, p.181-188, Como, Italy, 2000.

[26] Ciancarini, P., Tolksdorf R., and Zambonelli, F. A survey of coordination middleware for XML-centric applications. In *The knowledge engineering review*, Vol. 12., No 4., p.389-405, 2002.

[27] Mascolo, C., Capra, L., and Zachariadis, S., and Emmerich, W. XMIDDLE: A data-sharing middleware for mobile computing. *Wireless personal communication*, Vol. 21, No. 1, p.77-103, 2002.