XDependency: maintaining relationships between XML data resources

Dawn G. Gregg*

The Business School, University of Colorado at Denver, P.O. Box 173364, Campus Box 165, Denver, CO 80217-3364, USA

Received 14 October 2002; revised 30 March 2003; accepted 28 April 2003

Abstract

This paper proposes an XDependency description model and language that can be used as part of a distributed integrity control system for XML data resources. The dependency description language allows relationships to be specified between XML data available from different organizations. The paper also includes a prototype agent-based system capable of enforcing distributed dependency relationships. An XDependency maintenance system should make it possible to automate the distributed integrity control for XML resources.

Keywords: Document management; Integrity constraints; Distributed integrity control; XML

1. Introduction

XML is revolutionizing the way data is shared between applications and organizations [15]. As its use increases, the need for standards and tools that can be used to maintain and manipulate XML data grows. Several standards for working with XML data have been introduced and many more will be introduced in the coming years. These standards specify how XML documents can be constructed, viewed, linked, and queried [4,7,12]. These standards and ongoing research should allow individual XML data sources to be used in a manner similar to traditional relational databases.

One important problem that has not been adequately addressed for XML data is how to construct a distributed integrity control system that can be used to help ensure that the XML data maintains a desired level of data quality. As the use of XML expands, organizations will find that relationships exist between their XML data and XML data available from other organizations. It is necessary to develop standards that can connect these XML data sources such that data consistency can be maintained with respect to any Web data source, regardless of who ‘owns’ the base source for the data. This will move the Web one step closer to becoming a 'Semantic Web,' a global web of interrelated computer readable data [3].

Maintaining data consistency for distributed heterogeneous data resources requires methods to specify and enforce rules that describe associations between data or the required properties of the data [20]. This research addresses the problem of defining and storing integrity assertions in depth. It proposes an XML dependency description model and language that allows precise definition of the dependencies that can be specified between XML documents, even documents that are on remote systems and are controlled by different organizations. This dependency descriptor language (XDependency) is used to specify the dependency between related XML objects, the levels of permitted inconsistency, and the methods that can be used to restore the consistency if it is violated beyond the limits specified for the XML object.

The research also addresses issues related to enforcement of integrity assertions by developing a prototype agent based system that is used to implement a XML dependency maintenance system. The agent based system monitors XML resources, determines if updates to XML content are necessary, and performs those updates.

The rest of this article is organized as follows: Section 2 discusses related work on XML standards and techniques to monitor changes to documents found on the Web; Section 3 discusses issues related to distributed integrity control and why they are of increasing importance on the Web; Section 4 presents the proposed XDependency dependency description model, and Section 5 presents the prototype agent based system used to enforce distributed integrity constraints for XML documents.
2. Related work

Locating and extracting XML content has been the subject of ongoing research in the XML community for the past several years [23]. This has resulted in the development of the XPath, XPointer, and XLink standards. XPath is a language for addressing parts of an XML document and is designed to be used by both XSLT and XPointer [9]. XPointer, which is based on XPath, supports addressing into the internal structures of XML documents. It allows for examination of a hierarchical document structure and can identify its internal parts based on various properties, such as element types, attribute values, character content, and relative position [11]. XLink allows elements to be inserted into XML documents in order to create and describe links between resources [12]. In addition to these three standards, the World Wide Web Consortium is currently developing an XML query language, XQuery. XQuery is designed to provide flexible query facilities to extract data from real and virtual XML documents [4]. Research is also in progress to extend the current draft of XQuery to allow for update queries [26]. The XML dependency descriptors proposed in this research utilize XPointer to specify the location of specic XML content within XML documents. While other languages can be used (e.g. XQuery), XPointer was chosen for this application because it provides a concise format for selecting specific XML content. It also can be used to point to content within an XHTML Web document, which greatly expands the types of documents that can be related using XDependency.

Another area of active research has been on techniques to monitor changes to documents found on the Web. Within the database community research has been done on events and active rules that can be used to support integrity constraints, or maintain views of data [14]. These concepts are now being adapted to the Web domain. The Xylene project is investigating dynamic warehouses capable of storing a massive volume of XML data. Part of the research associated with this system has dealt with monitoring changes to XML data at the element level [10,19]. Other work has dealt with the development of triggers and active rules to help monitor changes to remote XML data and notify interested information consumers [6]. The XDependency model proposed in our research also uses events to detect changes to remote XML or XHTML documents so that data consistency can be maintained.

To date much of the XML research has focused on monitoring, querying and updating XML data [4,26] and not on maintaining the consistency of XML data resources. Alagic has proposed one of the few frameworks for specifying constraints for typed, object-oriented XML, which includes integrity constraints [1]. This framework utilizes structural transformations to relate different data models and their associated constraints such that the integrity of the data resource is preserved [1]. Other research utilizes active rules to enforce constraints that specify required properties of XML data [5]. However, so far, no mechanism exists to define and enforce integrity constraints that describe the associations between distributed heterogeneous XML data resources. Research related to the specification of distributed integrity constraints is described in detail in Section 3.

3. Distributed integrity control

An important problem in any data environment is how to ensure that the data maintains a desired level of data quality. That is, how can data be maintained such that it is accurate (recorded value in agreement with the actual value), timely (recorded value is not out of date), complete (all values for certain variables are recorded), and consistent (the representation of the data value is complete in all cases) [2,17]. Maintaining data quality in a data resource requires methods to specify and enforce rules that describe associations between data or the required properties of the data [20]. These are known as integrity constraints.

Integrity control has been investigated extensively in the database, distributed database, and data warehouse environments. Traditionally, integrity constraints are expressed declaratively to avoid problems of program data dependency, code redundancy, and poor performance of the procedural methods [20]. Florentin [13] was the first to suggest that integrity constraints be expressed using assertions of predicate calculus. This allows one to declare and modify complex integrity assertions. The XDependency model proposed as part of this research uses these traditional database approaches for expressing integrity constraints.

Distributed integrity control differs from that found in individual databases in that the constraints are defined for “data objects related by consistency requirements and possibly managed by different systems” [18, p.337]. XML data adds an additional difference because it can reside on remote systems controlled by different organizational entities. The main problem of distributed integrity control is that the communication and processing costs of enforcing distributed assertions can be prohibitive [20]. To address this problem, dependency descriptors were introduced to allow distributed database designers to specify under what conditions constraints needed to be enforced [22]. Dependency descriptors offer a framework for defining object dependencies and their representations. They are equivalent to integrity constraints except that they do not assume that consistency must be maintained at all times. Instead, dependency descriptors introduce a consistency requirement that specifies the levels of permitted inconsistency allowable for the given application [18,22]. This allows the administrator for the data to choose the events that will trigger the propagation of an update across the distributed system (e.g. a time interval between updates or to specific changes to the data). Dependency descriptors were introduced to reduce the system resources required to enforce
distributed dependency assertions and are applicable in application domains where data consistency is not required 100% of the time (e.g. data warehouses [24] and the XML domain).

Two major problems with defining a distributed integrity control system for heterogeneous data resources are determining how to define and store the integrity assertions, and how to enforce these assertions [20]. Section 4 addresses the first problem in depth. It proposes an XML dependency description model (XDependency) that allows precise definition of the dependencies that can be specified between XML documents, even documents that are on remote systems and are controlled by different organizations.

4. The XDependency dependency description model

XML dependency descriptors extend existing distributed integrity control methodologies for heterogeneous databases [18,22]. Under the proposed XDependency model, appropriate values for target XML elements can be defined using a dependency descriptor. This dependency descriptor is used to specify the dependency between related XML objects, the levels of permitted inconsistency, and the methods that can be used to restore the consistency if it is violated beyond the limits specified for the XML object. The following 4-tuple is proposed to describe XML object dependency:

\[ T = (S, P, C, A) \]

where

- \( T \) is the target XML object: the XML element (or elements) being maintained using the XDependency descriptor.
- \( S \) is the source XML objects: the associated base sources used to construct the target object’s value.
- \( P \) is the dependency predicate: a boolean-valued expression that specifies the relationship that should hold between the source Web objects and the target XML object.
- \( C \) is the consistency requirement: a boolean-valued expression that specifies when \( P \) must be satisfied.
- \( A \) is the action component: the consistency restoration procedures that should be activated to maintain or restore consistency.

The XDependency descriptors are expressed in a valid XML document. An XDependency description document contains multiple dependency assertions for a single XML document. Fig. 1 contains a Document Type Definition (DTD) for an XML document used to present the dependencies for a given Web document. This DTD defines how the XML tags should be interpreted by the application parsing the document [7]. This XML DTD specification should be appropriate for all Web-based dependency descriptors. The root element of the DTD is the dependency element. The dependency element has one required attribute: the Uniform Resource Locator (URL). This URL is used to specify the location of the Web document that the dependency descriptor describes constraints for. The dependency element contains one or more allowed instances of the nested element target. The target element contains one or more allowed instances of four nested elements: source elements, predicate elements, consistency elements and action elements. These are described in detail in the following sections.

4.1. Specifying data objects

A Web space consists of a set of documents, each of which exists at a unique location in the global/local Web continuum. This location is specified using a URL. Within a document, specific internal content can also be uniquely addressed based on properties like XHTML/XML tags, attribute values, character content, and relative position. The method of uniquely addressing the internal structure of a Web document will differ depending on the format and type of Web document that is being identified. The XML dependency description model proposes using XPointer to specify locations within Web documents. XPointer was chosen because it can also be used to point to content within an XHTML Web document, this greatly expands the number of documents XDependency descriptors can potentially define relationships for.

```xml
<?xml version="1.0"?>
<!DOCTYPE dependency [ 
<!ELEMENT dependency (target+)> 
<!ATTLIST dependency URL CDATA #REQUIRED> 
<!ELEMENT target (source+, predicate+, consistency+, action+)> 
<!ATTLIST target id CDATA #REQUIRED> 
<!ATTLIST target location CDATA #REQUIRED> 
<!ATTLIST target active CDATA #IMPLIED> 
<!ELEMENT source (#PCDATA)> 
<!ELEMENT predicate (#PCDATA)> 
<!ELEMENT consistency (#PCDATA)> 
<!ELEMENT action (#PCDATA)> ]>
```

Fig. 1. XDependency DTD.
XPointer is a language that allows pointing to actual content of an XML document, even when the original author of the target document did not provide fragment identifiers. XPointer relies on XML’s natural tree structure to allow specification of a set of criteria that selects a single element or item of content, or several such elements or items [9,11]. In general a location step has three parts: the axis, the node-test, and an optional predicate in this form:

\[
\text{axis} \,:\, \text{node-test}[\text{predicate}]\]

For example, if a XML designer wanted to maintain the consistency of a single price element (the target) within an XML document, it could be referenced as follows:

\[
\text{(target id = “I” location = “#xpointer(id(“wf-4753”))/child :: price”)}\]

This XPointer is defining a target element within the XML document that the dependency descriptor is defined for. To specify the location of the base data source, the XPointer is coupled with a URL:

\[
\text{http://www.xyz.com/p.xml\#xpointer(child :: prod[position() < 5]/child :: name)}\]

This source element specifies a set of four source data elements (the name elements found in the first four prod objects) selected from the document: http://www.xyz.com/p.xml.

4.2. Specifying dependency predicates

The dependency predicate specifies how to calculate a target XML object \(T\) value. It can be defined as:

\[
T(op)Expression
\]

where \((op)\) is an operator indicating the relationship between a target XML object and a set of source XML objects included in the Expression. Predicate expressions can contain arithmetic operations, logical operations, set operations, composition operations, aggregation operations, etc. [25]. There are also a number of dependency predicates that can be used to manipulate text-based content. Appendix A lists the dependency predicate functions that currently defined in for the XDependency System. These are similar to predicate expressions defined in the database and data warehouse domains [24].

For example, if the price listed on a company’s Web page is dependent on two suppliers’ prices multiplied by a corporate profit margin (50%) the dependency predicate for that data item could be expressed as follows:

\[
\text{(predicate) } \$\text{Target} = \$\text{Source[1]} \times \$\text{Source[2]} \times 1.5(/\text{predicate})
\]

where \(\$\text{Target}\) is a variable representing the target being maintained and \(\$\text{Source[1]}\) and \(\$\text{Source[2]}\) are variables representing the first and second sources for the given target.

It is possible that two or more data sources might independently yet identically populate a given target Web object’s value. In these situations, instead of defining separate dependency descriptors for the same Web object, a single dependency descriptor that defines when to use a given data source should be constructed. This requires the creation of multiple dependency predicates for a single target Web object. One example of a dependency descriptor using multiple sources to independently construct a single target Web object would be if the current stock price listed on a company’s Web page could be retrieved from either nasdaq.com or marketguide.com. The dependency predicate for that data item could be expressed as follows:

\[
\text{(predicate) } \text{target} = (\text{source[1]})/(\text{predicate}) \\
\text{(predicate) } \text{target} = (\text{source[2]})/(\text{predicate})
\]

In this situation, it is necessary to use separate consistency requirements and action procedures to define when predicate[1] should be used to construct the target object value and when predicate[2] should be used.

4.3. Specifying consistency requirements

The consistency requirement specifies the consistency constraints associated with the XML object. Using consistency constraints allows the XML data manager to specify the level of inconsistency that can be tolerated between the target XML object and the source XML objects or the frequency at which consistency needs to be restored. XML objects can be in one of three states: consistent, tolerated, or inconsistent [24]. This depends on the dependency predicate and consistency requirements as defined for each target object. If the dependency predicate is violated due to an update of the source XML page then the target XML object is no longer consistent with its sources. If this occurs and the consistency requirements are not violated, then the target object moves into a tolerated state. If both the dependency predicate and the consistency requirements are violated, then the target XML object is in an inconsistent state and the appropriate actions are triggered [24].

The consistency predicate consists of a SOAP message that specifies what methods to use to monitor a given source (where SOAP is Simple Object Access Protocol [16]). The SOAP message also specifies events that determine the point of restoration for the XML content. The consistency requirements are specified using events where the event is ‘some happening of interest’ [14]. Events can be classified as:

- **Temporal events.** These events specify a point in time or interval at which the related data items must be consistent [14].
A complete list of the temporal, mutation and data state events that are supported using the XDependency language are found in Appendix B.

It is possible for mutation events (insertion, deletion, or update operations) occurring in remote XML data sources to be propagated to target Web sites. This would be accomplished using Web services that allow registration of event listeners on the source XML data (similar to the push service described in Ref. [6]). Web services are ideal for this application because they support loosely coupled service-oriented architectures that allow clients to contact them (e.g. register to receive specific update information) using their contact information [8]. These Web services rely on two primary protocols: SOAP, a lightweight protocol for exchange of information in a decentralized, distributed environment [16] and the Document Object Model Level 2 specification which provides a generic event system for the Web. Under DOM, event listeners associated with DOM nodes detect events occurring on the nodes to which they are associated or on their descendents [21].

If the source data site does not support the use of Web Services to push events to remote sites, it is still possible to detect changes to the source content, utilizing agents that monitor the source Web site. Since changes to source data objects in this environment do not automatically trigger transaction events, these XML consistency requirements need to incorporate a temporal event that defines when (how often) the intelligent agent should verify the consistency of the XML objects.

The following are examples of two possible consistency requirement definitions. The first consistency requirement assumes that the first source (source[1]) does not provide an event listening Web service. This consistency requirement calls a verifySource method that will periodically check to see if specific events have occurred. The second consistency requirement assumes the second source (source[2]) provides an event listening service. It calls a listenSource method that will add an appropriate listener to that source:

\[
\text{(consistency)} \quad \text{(source)}\text{source[1]} / \text{source[2]}
\]

\[
\text{(consistency)} \quad \text{soap:Envelope xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/"}
\quad \text{soap:encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"}
\quad \text{soap:Body}
\quad \text{m:verifySource xmlns:m = "http://132.194.39.152/services"}
\quad \text{source}\text{source[2]}/\text{source[1]}
\]

The method calls are placed inside a standard SOAP envelope. They pass the source to be verified and the event(s) that must be triggered to cause the consistency requirement to return true. The variables source[1] and source[2] will be replaced with the full source elements prior to submitting the method call. Each of the above consistency requirements could result in a different action procedure being invoked.

4.4. Specifying action procedures

The action procedure specifies a SOAP method that will be executed to restore the consistency of a target XML object when one of its consistency requirements returns true. In general many action procedures can be associated with a single dependency descriptor. Each action procedure can contain single or multiple procedures for restoring the consistency. The restoration procedure can be specified as

\[
\text{Condition}_i \wedge r_i(\text{arg}_i)
\]

where \text{Condition}_i is a logical expression that defines the restoration point of the XML object by the procedure \text{r}_i, and where \text{r}_i can reference any SOAP method.

In traditional databases and data warehouses, action procedures always represent a consistency restoration procedure that restores the consistency of the target XML object. In the XDependency system this type of action procedure is implemented using a SOAP method updateTarget. Web-based systems differ from traditional, single organizational systems because source XML objects may change dramatically or disappear. In these situations, the SOAP methods notify or deactivateTarget may be used. The SOAP notify method records the change detected in the source content so that a human can evaluate the
change and determine how it effects the target XML object. The SOAP deactivateTarget method would allow specific dependency descriptors to be ‘turned off’ if they are no longer valid.

The following is a sample activation procedure that calls the SOAP updateTarget method:

```xml
<action>$consistency[1] ::
  <soap:Envelope xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
  >
    <soap:Body>
      <m:updateTarget xmlns:m = "http://132.194.39.152/services">
        <url>$dependency/@URL</url>
      </m:updateTarget>
    </soap:Body>
  </soap:Envelope>
</action>
```

The action procedure would be invoked if the first consistency requirement for the current target ($consistency[1]) returns true (meaning the first source object may have been changed). It passes the URL for the dependency descriptor object and the entire target object to a SOAP updateTarget method. This allows the method to update the specific target object based on the dependency predicate and the appropriate base source data.

5. XDependency implementation example

Once a language for defining and storing XML integrity assertions has been created, the other major problem that needs to be addressed is the enforcement of the XDependency constraints. This section addresses issues related to the enforcement of XDependency integrity assertions through the development of prototype agent-based distributed integrity control system. This section also includes a discussion of additional design issues that must be addressed in any distributed integrity control system for XML data.

5.1. Example XDependency implementation

Fig. 2 shows an example of a prototype agent-based XDependency maintenance system. The prototype system consists of two different agents. The Dependency Manager Agent is responsible for maintaining a list of XML pages for which XDependency constraints have been specified and for updating the XML page if the source-verification agent indicates the source data has changed beyond what is tolerated. The Source Verification Agent is responsible for checking the current values of the XML data sources and initiating an update procedure if the distributed integrity constraints are violated. The agents are deployed as Web Services and communicate via SOAP messages. They are ‘loosely coupled,’ functioning as two largely independent applications. The prototype system is purely a proof-of-concept system and does not contain all of the capabilities a full XDependency maintenance system will need to implement.

![Fig. 2. Sample XDependency maintenance system.](image-url)
A prototype XDependency system has been implemented in Java and has been tested using the example scenarios including the one described in Appendix C. In the prototype system, the agents are not mobile, however, the dependency descriptors themselves remain distributed. When a dependency descriptor is created (or updated) it is submitted to a dependency-manager agent using a SOAP message like the one below (Step 1 in Fig. 2):

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:encoding="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:m="http://132.194.39.152/services">
  <soap:Body>
    <m:addDependency xmlns:m="http://132.194.39.152/services">
      <url>http://carbon.cudenver.edu/~dgregg/XML/cb_dep.xml</url>
    </m:addDependency>
  </soap:Body>
</soap:Envelope>
```

The `addDependency` method opens the XDependency descriptor document. It extracts the consistency requirements for each target and replaces any embedded variables (e.g. $source[1]) with their actual values. The dependency-manager agent would then add a unique ID to the header of the SOAP message (for tracking purposes) and send the complete SOAP messages to the appropriate source-verification agent (Step 2 in Fig. 2). The modified SOAP message might look like the one below:

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:encoding="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:m="http://132.194.39.152/services">
  <soap:Header>
    <xdep:MessageHeader xmlns:xdep="http://132.194.39.152/services">
      <consistencyID id="975"/>
    </xdep:MessageHeader>
  </soap:Header>
  <soap:Body>
    <m:verifySource xmlns:m="http://132.194.39.152/services">
      <source>http://www.abc.com/prod.xml#xpointer(id="43")/child::msrp</source>
      <frequency>every(time(HR = 24))</frequency>
      <event>deleted</event>
    </m:verifySource>
  </soap:Body>
</soap:Envelope>
```

If a message is submitted to the `verifySource` method, the source-verification agent will add the source to the list of sources that are being monitored. The original value(s) for the source will be retrieved and used to compare subsequent values to. The source document is then added to a queue so that source document will be retrieved and verified at the appropriate times (Step 3 in Fig. 2). If the source-verification agent detects that the source content being monitored was updated or deleted it will submit an event `method` call containing the event type, the original value(s) and the updated value(s) to the originating dependency-manager agent (Step 4 in Fig. 2).

Once the dependency-manager agent receives notification that an event of interest has occurred, it is responsible for invoking the appropriate action for the given condition (Step 5 in Fig. 2). It retrieves the target’s XDependency descriptor document. It then extracts the action procedures for the triggered consistency requirement and replaces any embedded variables (e.g. $source[1]) with their actual values. The dependency-manager agent would send the complete SOAP messages to the appropriate action procedure (Step 6 in Fig. 2). Depending on the procedure invoked either the target content would be updated (`updateTarget`), or the appropriate human would be notified of the event via email or an error log (`notify`).

5.2. Agent design issues

One design issue that exists in any implementation of the XDependency system is the possibility of circular references [20]. This situation might occur if two or more Web objects reference each other and if these references are not the inverse of each other. For example, a supplier could define a dependency descriptor that sets the supplier’s price to be 85% of the vendor’s price and the vendor could define a dependency descriptor that sets the vendor’s price to be 125% of the supplier’s price. In this situation, each time the vendor resets its price the supplier’s price will be inconsistent and each time the supplier resets its price the vendor’s price will be inconsistent. This cycle is said to be unstable because it causes an infinite number of updates.

One way to avoid unstable cycles is to verify that each new dependency descriptor that is added does not conflict with an existing dependency descriptor. However, an additional problem is created in the Web domain because a single organization may not control all of the Web documents for which dependency descriptors are defined. Thus, even if all new dependency descriptors you define do not result in unstable cycles when they are defined, it is still possible for an unstable cycle to be introduced as the result of a dependency descriptor defined by an individual outside your organization.

It is recommended that consistency requirements only be executed at specific time intervals so that if unstable cycles are introduced the Web site will only be updated at the
specified times (not infinitely). It is also recommended that agents also periodically check dependency paths to verify that no unstable cycles exist. One way to recognize cycles in a directed graph is to do a depth-first search and check for edges that reach vertices that have already been reached.

A second issue that needs to be addressed in the design of an XDependency system is the issue of scalability. As the number of conditions being monitored becomes large it is possible that the system could become inefficient and unmanageable. Two alternate approaches can be incorporated into a fully functional XDependency system. First, approaches that minimize the number of document retrievals can be incorporated into the final system design. And, second, since dependency descriptors are distributed XML documents, the monitoring agents can also be distributed and limit the number of conditions monitored to an optimal number.

A final issue is that the XDependency system is designed to allow relationships between XML documents from different organizations. This creates security concerns for organizations. For sites installing dependency descriptors, the primary issue will be whether or not to allow the automatic updating of Web content by the system. To allow automatic updating, write access must be given to the agent, possibly opening the system to malicious attacks. In addition, automatic updating also presents the possibility of corrupting the target XML data source. These risks are characteristics of any agent-based mechanism. However, they can be addressed by only allowing agents from trusted sites update access to your site. Dependency constraints can also be designed such that values outside of a specified range do not trigger an update to the target data (they could trigger a notification action instead). Thus, it is possible to minimize the security risks associated with automatic updates.

Source data sites can also have security concerns. They need to consider whether they will allow access to their XML data resources. They must decide whether to allow event listeners to watch for events on their site, whether to allow intelligent agents to monitor their site data, or whether they want to establish a robot exclusion policy that prohibits any automated monitoring of their site data.

While there are issues that may make XDependency descriptors inappropriate for some applications, there are numerous situations for which automated update of XML data would be potentially beneficial. XDependency descriptors provide a new capability that could play an important role in future Web-based infrastructures.

6. Conclusions

This paper proposed XDependency, an XML-based language for expressing dependencies between XML documents, and a prototype agent-based application that can be used to enforce them. The XDependency distributed integrity control system allows the specification and maintenance of distributed integrity constraints for XML resources. An XDependency maintenance system will make it possible to automate many of the processes necessary to update XML-based information automatically.

The XDependency language has been implemented for several data sets. However, to demonstrate the viability of this concept it will be necessary to validate that this model meets the needs of a wide variety of organizations. This will involve either extensive experimental validation of the distributed dependency descriptors or case studies that show the implementation of dependency description systems in different domains.

This research included the development of a prototype agent-based system that is capable of interpreting and acting on the XDependency descriptors. However, the capabilities of this system will need to be expanded if XDependency is to become a reliable distributed integrity control system. The design issues discussed in Section 5.2 will need to be addressed further and performance analyses for these systems will need to be conducted. In addition, it will be necessary to develop tools to assist Web developers with the task of developing and maintaining the dependency descriptors for their Web sites.

Appendix A. Dependency predicate expressions supported under XDependency

<table>
<thead>
<tr>
<th>Tables A1 and A2</th>
</tr>
</thead>
</table>

Table A1

<table>
<thead>
<tr>
<th>Predicate expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source[1] + source[2]</td>
<td>Source Web objects can be combined using arithmetic operators (+, -, *, /, %)</td>
</tr>
<tr>
<td>Sum (source[1], source[2])</td>
<td>The summation of its sources source[1], etc. where individual sources could contain multiple data items</td>
</tr>
<tr>
<td>Average (source[1], source[2])</td>
<td>The average of its sources source[1],...,source[n]</td>
</tr>
<tr>
<td>Median (source[1], source[2])</td>
<td>The median of its sources: source[1],...,source[n]</td>
</tr>
<tr>
<td>Mode (source[1], source[2])</td>
<td>The mode of its sources source[1],...,source[n]</td>
</tr>
<tr>
<td>Max(source[1], source[2])</td>
<td>The maximum of its sources source[1],...,source[n]</td>
</tr>
<tr>
<td>Min(source[1], source[2])</td>
<td>The minimum of its sources source[1],...,source[n]</td>
</tr>
<tr>
<td>Product (source[1], source[2])</td>
<td>The product of its sources source[1],...,source[n]</td>
</tr>
<tr>
<td>Format (source[1],2)</td>
<td>Formats a floating point number to two decimal places</td>
</tr>
<tr>
<td>Int (source[1])</td>
<td>Rounds a number down to the nearest integer</td>
</tr>
</tbody>
</table>
Appendix B. Events supported under XDependency

Tables B1–B3

Appendix C. Example

Consider an XML Web page for a company that sells imported coffee. Fig. C1 shows an example XML page for this fictitious firm. It contains the prices for coffees obtained from different sources. The only difference between the XML source for this Web page and that for a typical XML document is that it has an optional tag specifying it uses an
XDependency descriptor:

```xml
<dependency href="cb_dep.xml"/>
```

The XDependency descriptor for this Web page is found at http://carbon.cudenver.edu/~dgregg/XML/cb_dep.xml. In this example, the price for the Malaysian Coffee is being maintained with a dependency descriptor (Fig. C2).

Fig. C2 shows a sample dependency descriptor for this XML Web object. The target XML object is dependent on data from a single XML source. It has multiple consistency requirements and action procedures. The initial XML tag:

```xml
<dependency URL="http://carbon.cudenver.edu/~dgregg/XML/cb.xml"/>
```


Fig. C2. Sample dependency descriptor.
The dependency descriptor indicates that the first (and only) target being maintained is the price associated with the XML object with id ‘wf43’ (the first price in the table in Fig. C1). This target is dependent on a single source msrp element found at the URL: http://cob-dell1.cudenver.edu:81/dawn6240/prod.xml. The predicate element in Fig. C2 indicates that the target value is equal to this source msrp value divided by 16 and multiplied times 2. This is because the source price is for 16 units and the ‘Ground Coffee’ reseller wants to multiply the price for one unit by a profit margin of 2.

There are two consistency requirements specified for the Malaysian Coffee dependency descriptor. The first consistency requirement returns true if source[1] is updated. The second consistency requirement returns true if source[1] is deleted. Both consistency requirements are maintained using the SOAP verifySource method, which will check source[1] every 24 h.

The two action procedures in Fig. C2 perform either update or notification procedures depending on the consistency requirements that trigger them. The first action procedure is triggered if consistency[1] returns true. It calls the SOAP updateTarget method and passes it the URL for the Web page to be updated and the complete target element to be used in constructing the updated value. The second action procedure is invoked if consistency[2] returns true. It uses the SOAP notify method to notify the appropriate human that the base source for the data was deleted.

This dependency descriptor can be maintained utilizing the prototype XDependency agent system described in Section 5.1.

References