HIERARCHICAL VS. RELATIONAL XML SCHEMA DESIGNS

A STUDY FOR THE ENVIRONMENTAL COUNCIL OF STATES

REPORT ECO41T1

Neil A. Chaudhuri
Jessica L. Glace
Gregory A. Wilson

LMI
GOVERNMENT CONSULTING

JUNE 2006

Contents

Figures

Table

Hierarchical vs. Relational XML Schema Designs

INTRODUCTION

The Environmental Information Exchange Network—a partnership between the Environmental Protection Agency (EPA) at the federal level and environmental departments at the state level—has been working for more than 5 years to develop a network for exchanging data electronically between the two parties. One function of the Exchange Network is to enable the flow of data using Extensible Markup Language (XML). These XML data flows must comply with both XML Schema, the specification recommended by the World Wide Web Consortium (W3C), and the XML design rules and conventions established for the Exchange Network. The latter provide guidance to schema developers across the Exchange Network and are intended to ensure that the schemas are developed consistently.

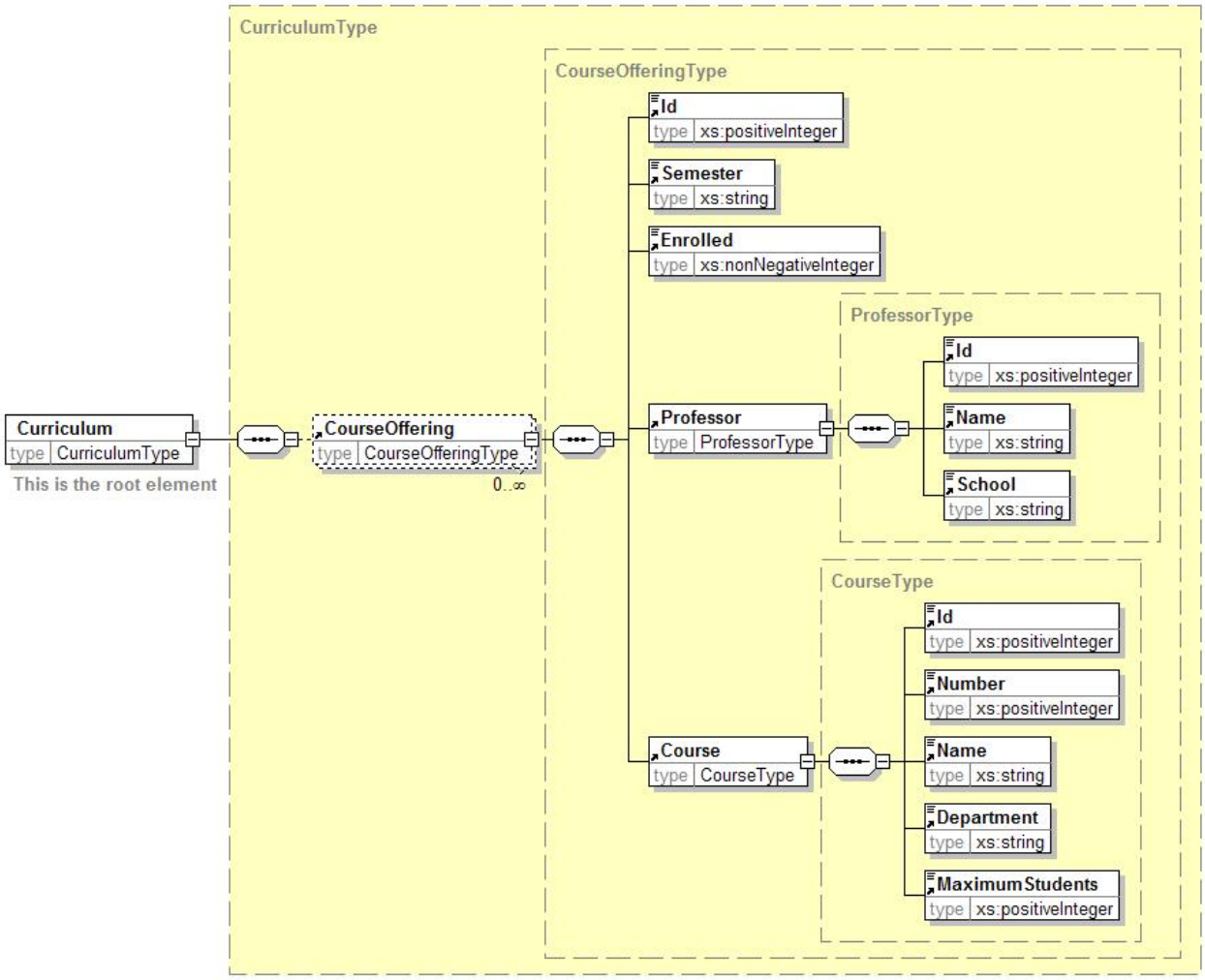
In a recent data flow development effort with EPA's Office of Solid Waste, two different developers created vastly different schemas for the same XML data flow. One developer used a schema design approach known as *hierarchical*, while the other used a schema design approach known as *relational*. Both, however, complied with the W3C XML Schema specification and the Exchange Network's XML design rules and conventions.

To ensure that the design of schemas for XML data flows on the Exchange Network are consistent, additional policies are required. Ideally, the Exchange Network needs to determine whether hierarchical or relational designs should be used. To that end, this report defines the two types of XML schema structural designs: hierarchical and relational. It then evaluates the two designs in terms of how well they satisfy those criteria deemed most pertinent to the Exchange Network. The results of that analysis can be used to provide a basis to promote consistent design of Exchange Network XML schema documents in a decentralized environment.

HIERARCHICAL SCHEMA DESIGN

The term "hierarchical" refers to schemas that require instances to maintain relationships among entities strictly through nested structures. Relationships among elements are defined solely by their relative physical locations in the XML instance. Figure 1 is an XML schema fragment illustrating such relationships. Specifically, it shows the many-to-many relationship between a professor and a course as conveyed in a hierarchical schema of course offerings.

Figure 1. Hierarchical Schema Tree Diagram



Within Curriculum, a CourseOffering occurs zero to an infinite amount of times. Each CourseOffering has the following elements:

- *Id*—identifies the *CourseOffering* uniquely. Id is required once for each iteration of CourseOffering.
- *Semester*—provides the semester in which a course with a particular professor is offered (e.g., Spring 2006, Fall 2005, Summer 2005). Semester is required once for each iteration of *CourseOffering*.
- *Enrolled*—provides the number of students currently enrolled. Enrolled is required once for each iteration of *CourseOffering*.
- *Professor*—required once for each iteration of *CourseOffering* and contains the following elements:
  - *Id*—identifies the individual professor.

- *Name*—provides the full name of the professor.
- *School*—provides the full name of the school from which the professor graduated.

- *Course*—required once for each iteration of CourseOffering and contains the following elements:
  - *Id*—identifies the individual course.
  - *Number*—provides the number of the course.
  - *Name*—provides the full name of the course.
  - *Department*—provides the full name of the department.
  - *MaximumStudents*—identifies the maximum number of students allowed in a specific course.

The following XML text illustrates how this hierarchical design would appear in an XML instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<Curriculum xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ecos_h.xsd">
    <CourseOffering>
        <Id>1</Id>
        <Semester>Fall</Semester>
        <Enrolled>128</Enrolled>
        <Professor>
            <Id>1</Id>
            <Name>John Brown</Name>
            <School>UNC</School>
        </Professor>
        <Course>
            <Id>3</Id>
            <Number>150</Number>
            <Name>Biology I</Name>
            <Department>Health Sciences</Department>
            <MaximumStudents>150</MaximumStudents>
        </Course>
    </CourseOffering>
    <CourseOffering>
        <Id>2</Id>
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
        <Professor>
            <Id>1</Id>
            <Name>John Brown</Name>
            <School>UNC</School>
        </Professor>
        <Course>
            <Id>1</Id>
```

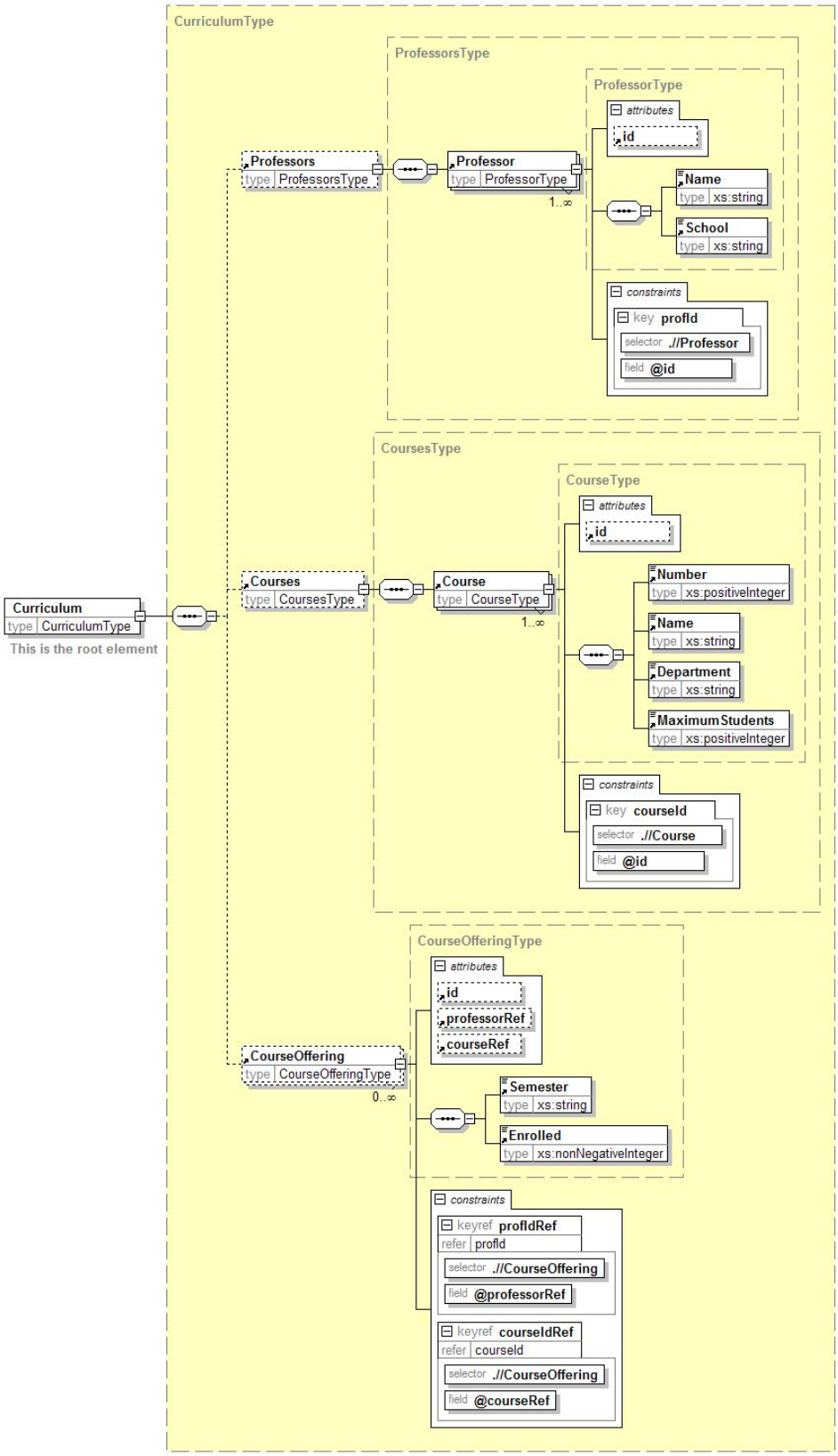
```
            <Number>114</Number>
            <Name>Computer Science I</Name>
            <Department>Computer Science</Department>
            <MaximumStudents>30</MaximumStudents>
        </Course>
    </CourseOffering>
    <CourseOffering>
        <Id>3</Id>
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
        <Professor>
            <Id>2</Id>
            <Name>Elanor Fisher</Name>
            <School>Harvard</School>
        </Professor>
        <Course>
            <Id>3</Id>
            <Number>150</Number>
            <Name>Biology I</Name>
            <Department>Health Sciences</Department>
            <MaximumStudents>150</MaximumStudents>
        </Course>
    </CourseOffering>
    <CourseOffering>
        <Id>4</Id>
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
        <Professor>
            <Id>2</Id>
            <Name>Elanor Fisher</Name>
            <School>Harvard</School>
        </Professor>
        <Course>
            <Id>4</Id>
            <Number>160</Number>
            <Name>Biology II</Name>
            <Department>Health Sciences</Department>
            <MaximumStudents>150</MaximumStudents>
        </Course>
    </CourseOffering>
    <CourseOffering>
        <Id>5</Id>
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
        <Professor>
            <Id>2</Id>
            <Name>Elanor Fisher</Name>
            <School>Harvard</School>
        </Professor>
        <Course>
            <Id>6</Id>
            <Number>200</Number>
            <Name>Calculus II</Name>
            <Department>Mathematics</Department>
            <MaximumStudents>200</MaximumStudents>
        </Course>
```

```
    </CourseOffering>
    <CourseOffering>
        <Id>6</Id>
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
        <Professor>
            <Id>3</Id>
            <Name>David Darwin</Name>
            <School>Yale</School>
        </Professor>
        <Course>
            <Id>6</Id>
            <Number>200</Number>
            <Name>Calculus II</Name>
            <Department>Mathematics</Department>
            <MaximumStudents>200</MaximumStudents>
        </Course>
    </CourseOffering>
    <CourseOffering>
        <Id>7</Id>
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
        <Professor>
            <Id>3</Id>
            <Name>David Darwin</Name>
            <School>Yale</School>
        </Professor>
        <Course>
            <Id>5</Id>
            <Number>100</Number>
            <Name>Calculus I</Name>
            <Department>Mathematics</Department>
            <MaximumStudents>200</MaximumStudents>
        </Course>
    </CourseOffering>
</Curriculum>
```

RELATIONAL SCHEMA DESIGN

XML schema documents that use a relational design maintain relationships among entities through the use of keys rather than physical nesting. Similar to the primary and foreign key relationships found in relational database tables, relationships in the XML instance are established through unique identifiers for elements and references to those in other elements. The physical location of an element is irrelevant. The XML Schema specification offers two mechanisms for establishing unique identifiers for element declarations and references to those identifiers. The preferred method is to use the *key* and *keyRef* elements because of their flexibility. These elements build on the *id* and *idref* functionality XML Schema inherited from XML Document Type Definitions (DTDs). Figure 2 illustrates the many-to-many relationship between a professor and a course as conveyed in a relational schema of course offerings.

Figure 2. Relational Schema Tree Diagram



Within Curriculum are three optional elements: *Professors*, *Courses*, and *CourseOffering*. Each is described below.

Professors occurs zero to once and contains the following:

- *Professor*—occurs once to an infinite number of times and contains the following:
  - *id* attribute—identifies the professor. Id is optional.
  - *Name* element—provides the full name of the professor. Name is required once for each iteration of professor.
  - *School* element—provides the full name of the school from which the professor graduated. School is required once for each iteration of professor.
  - *profid* key—uses the XML Schema *key* element to uniquely identify the professor so that it can be referenced by other elements.
    - .//Professor—provides the XPath expression that specifies the set of elements within which the values specified by the XML Schema *field* element must be unique.
    - @id—provides the attribute within the elements specified by the XML Schema field element (i.e., .//Professor) that uniquely identifies a professor.

Courses occurs zero to once and contains the following elements:

- *Course*—occurs once to an infinite number of times and contains the following:
  - *id* attribute—identifies the individual course. Id is optional.
  - *Number* element—provides the number of the course. Number is required once for each iteration of course.
  - *Name* element—provides the full name of the course. Name is required once for each iteration of course.
  - *Department* element—provides the full name of the department. Department is required once for each iteration of course.
  - *MaximumStudents* element—identifies the maximum number of students allowed in a specific course. MaximumStudents is required once for each iteration of course.

- *courseId* key—uses the XML Schema *key* element to uniquely identify the professor so that it can be referenced by other elements.
  - .//Course—provides the XPath expression that specifies the set of elements within which the values specified by the XML Schema *field* element must be unique.
  - @id—provides the attribute within the elements specified by the XML Schema field element (i.e., .//Course) that uniquely identifies a course.

CourseOffering pairs the identifiers for the appropriate *Professor* and *Course* elements to establish the relationship between them, instead of establishing a physical structure containing the entire *Professor* and *Course* elements.

CourseOffering contains the following:

- *id* attribute—identifies the individual course offering; id is optional.
- *professorRef* attribute—is a reference back to the profId attribute and identifies the professor to which this CourseOffering applies; professorRef is optional.
- *courseRef* attribute—is a reference back to the courseId attribute and identifies the course to which this CourseOffering applies; courseRef is optional.
- *Semester* element—identifies a specific semester; Semester is required once for each iteration of CourseOffering.
- *Enrolled* element—provides the number of students currently enrolled; Enrolled is required once for each iteration of CourseOffering.
- profIdRef keyref—uses the XML Schema *keyref* element to refer back to a previously defined key.
  - profId—identifies the key to reference (i.e., profId).
  - .//CourseOffering—provides the XPath expression that specifies the set of elements within which the values specified by the XML Schema *field* element must be unique.
  - @professorRef—provides the attribute within the elements specified by the XML Schema field element (i.e., .//CourseOffering) that uniquely references a professor.

- courseIdRef keyref—uses the XML Schema *keyref* element to refer back to a previously defined key.
  - courseId—identifies the key to reference (i.e., courseId).
  - .//CourseOffering—provides the XPath expression that specifies the set of elements within which the values specified by the XML Schema *field* element must be unique.
  - @courseRef—provides the attribute within the elements specified by the XML Schema field element (i.e., .//CourseOffering) that uniquely references a course.

The following XML text illustrates how this relational design would appear in an XML instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<Curriculum xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ecos_r.xsd">
    <Professors>
        <Professor id="1">
            <Name>John Brown</Name>
            <School>UNC</School>
        </Professor>
        <Professor id="2">
            <Name>Elanor Fisher</Name>
            <School>Harvard</School>
        </Professor>
        <Professor id="3">
            <Name>David Darwin</Name>
            <School>Yale</School>
        </Professor>
    </Professors>

    <Courses>
        <Course id="1">
            <Number>114</Number>
            <Name>Computer Science I</Name>
            <Department>Computer Science</Department>
            <MaximumStudents>30</MaximumStudents>
        </Course>
        <Course id="2" >
            <Number>214</Number>
            <Name>Computer Science II</Name>
            <Department>Computer Science</Department>
            <MaximumStudents>30</MaximumStudents>
        </Course>
        <Course id="3">
            <Number>150</Number>
            <Name>Biology I</Name>
            <Department>Health Sciences</Department>
            <MaximumStudents>150</MaximumStudents>
        </Course>
```

```
        <Course id="4">
            <Number>160</Number>
            <Name>Biology II</Name>
            <Department>Health Sciences</Department>
            <MaximumStudents>150</MaximumStudents>
        </Course>
        <Course id="5">
            <Number>100</Number>
            <Name>Calculus I</Name>
            <Department>Mathematics</Department>
            <MaximumStudents>200</MaximumStudents>
        </Course>
        <Course id="6">
            <Number>200</Number>
            <Name>Calculus II</Name>
            <Department>Mathematics</Department>
            <MaximumStudents>200</MaximumStudents>
        </Course>
    </Courses>

    <CourseOffering id="1" professorRef="1" courseRef="3">
        <Semester>Fall</Semester>
        <Enrolled>128</Enrolled>
    </CourseOffering>
    <CourseOffering id="2" professorRef="1" courseRef="1">
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
    </CourseOffering>
    <CourseOffering id="3" professorRef="2" courseRef="3">
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
    </CourseOffering>
    <CourseOffering id="4" professorRef="2" courseRef="4">
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
    </CourseOffering>
    <CourseOffering id="5" professorRef="2" courseRef="6">
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
    </CourseOffering>
    <CourseOffering id="6" professorRef="3" courseRef="6">
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
    </CourseOffering>
    <CourseOffering id="7" professorRef="3" courseRef="5">
        <Semester>Fall</Semester>
        <Enrolled>25</Enrolled>
    </CourseOffering>
</Curriculum>
```

EVALUATION

Both the hierarchical design and the relational design offer advantages and disadvantages. To determine which would be more appropriate for developing schemas for XML data flows on the Exchange Network, we evaluated how well each design type satisfied each of several criteria deemed of greatest significance to the Exchange Network. Those criteria, in order of importance, are data integrity, interoperability and loose coupling, message size, readability, processing ease, and established practices within the XML schema development community.

The following subsections define the criteria, discuss the ability of the two design types to meet the criteria, and then analyze the design types in terms of their advantages and disadvantages for the Exchange Network.

Data Integrity

Those who work with databases are well aware of the painstaking measures necessary to maintain data integrity, the preservation of the data and their relationships. Among the concerns with data integrity are the following:

- *Uniqueness*. This is the manner in which data elements are identified to distinguish them from other data elements.
- *Referential integrity*. This is the enforcement of parent-child relationships among data to ensure that no stray child data can exist without a corresponding parent. Therefore, if a parent is deleted, all associated children are deleted as well.
- *Anomalies*. In databases, "update," "insert," and "delete" anomalies are possible. These are breaches in integrity that may occur when data are denormalized. Delete anomalies, for example, occur when the deletion of one data element leads accidentally to the deletion of another data element.

With regard to XML schemas and XML instances, there are significant data integrity questions to consider. One is the manner in which a data element is deemed unique by processing logic. Another is the manner in which parent-child relationships among data are reflected in the instance. Finally, there is the manner in which normalization is achieved in order to reduce repetition of data and decrease the likelihood of any anomalies.

HIERARCHICAL DESIGN

In an XML instance that conforms to a hierarchical schema, there is no inherent mechanism for enforcing uniqueness of data. As a practical matter, some identification mechanism is necessary, and there are several approaches to this. One is to inject into each hierarchy primary key values from the database in which the

XML instance data originated. These keys are used to uniquely identify one or more XML data structures. This approach, however, does not guarantee key uniqueness throughout the XML schema. In other words, two elements in a hierarchical schema may have the same id associated with them, requiring the receiving application to determine whether they are duplicative. For example, an application receiving an instance with a 'Professor' element with an id of '1' and a 'Course' element with an id of '1' would need to verify that the two elements are conceptually distinct and not duplicative of one another. This mechanism demands a strategy for mapping keys to the corresponding keys in the destination database (where data from the instance will be stored). Yet the use of a database key spares application developers the need to develop business rules for generating a natural key and applying these rules throughout the document.

Normalization is not applicable to XML instances conforming to hierarchical schema. The consequences for data integrity within the instance depend on the manner in which it is processed. If the processing of the XML instance leaves it entirely intact before its information is stored in the database, then there are no consequences. However, if the instance were to be modified in some way—for example, through XSLT processing—then the integrity of the data could be jeopardized through update and delete anomalies.

Consider a many-to-many relationship in a database. In a normalized database, this relationship is represented with an association table between two entities. In a hierarchical XML instance, this relationship is represented with a collection of "container" elements composed of the two entities, each in its entirety. An update anomaly would occur in the instance if an update were made to an element within a container without the same update occurring to all equivalent elements within other containers. A delete anomaly would occur if somehow all container elements containing a common entity were lost, for then no record of that entity would exist. Changes to the data such as these would be all but undetectable if they occurred, for the instances would likely remain valid against the original XML schema.

Precisely because normalization is not applicable to an XML instance validated against a hierarchical schema, neither is referential integrity. Without normalization, there are no key-based parent-child relationships between entities to consider. Only those physical parent-child relationships inherent to hierarchical structures will be present.

RELATIONAL DESIGN

In an XML instance validated against a relational schema, use of the *key* and *keyref* attributes serves as the mechanism for establishing the uniqueness of an element over a node set as defined by an XPath expression. A significant advantage of this approach is that uniqueness is enforced by the associated schema since the instance will not pass validation if the keys are not unique. As is the case with identifiers found in their hierarchical counterparts, the key values in rela-

tional instances may expose key information from the source database. Also, it may be difficult for a receiving application to map source-database keys to the respective records in the destination database.

Normalization is accomplished by an instance validated against a relational schema in a manner virtually identical to that in a relational database. The data elements in an instance are normalized by presenting entities identified through the *key* attribute in a single location and then referencing those throughout the instance where applicable and valid by means of the *keyRef* attribute. This strategy is particularly effective for reducing the likelihood of anomalies if changes are made to the instance. Update anomalies are averted because entities appear only once in the document; therefore, an update to the entity is reflected throughout the document through *keyRef*. Delete anomalies are averted because entities remain distinct from one another; therefore, deleting a relationship will never result in the deletion of an entire entity.

Referential integrity in instances validated against a relational schema is enforced by the schema itself. According to the XML Schema specification, all *keyRef* attribute values in an instance must match *key* attribute values (over the same node set) found in the instance. Therefore, relational instances ensure referential integrity—yet only in theory. It is conceivable that more forgiving validators will not vigorously enforce XML Schema constraints. In such cases, referential integrity may be compromised in practice.

ANALYSIS

Referential integrity is of concern to Exchange Network stakeholders for two forms of communication. The first is state-to-state communication. States receiving XML instances are unlikely to modify these documents significantly. In such cases, a hierarchical approach is beneficial, for referential integrity will not be compromised. Because states are only reading the data, they need only apply basic transformations to the instances in order to generate instances better suited to their platforms.

The second case is state-to-federal communication. In this case, each XML instance will be transformed extensively by application software at the federal level. Consequently, data integrity becomes a major concern, for anomalies become far more likely to occur. In this case, a relational approach is beneficial. By normalizing the data in the instance, transactions performed on a document received by a federal agency will not compromise data integrity. Transformation mechanisms will need to be more complex than their hierarchical counterparts because of more complicated XPath functions and expressions. Nevertheless, data integrity should remain reliable.

Interoperability and Loose Coupling

As individual organizations develop their own software architectures to meet their business objectives, these architectures must communicate effectively and efficiently with those of their trading partners throughout the Exchange Network. The challenge of interoperability is formidable given the infinite permutations of technologies and platforms that constitute systems, yet it is imperative that interoperability be achieved to the greatest extent possible in order to yield the greatest value.

Interoperability is achieved through loose coupling. Loose coupling is a principle whereby a layer of abstraction is placed between modules in a system so that changes to one module do not influence others that depend on it. The abstraction, which is often based on some standard, shields the technical details of the underlying module from its clients. Interoperability is then achieved; all that is necessary to communicate effectively is knowledge of the abstraction standard, rather than knowledge of any particular technology or platform.

With regard to XML, the issue of interoperability is a function of the abstraction provided between XML instances and the technologies that produce and consume them. These include databases primarily but also technologies—such as programming code and XSLT—that are used to process instances.

Abstraction is achieved through standards. However the data are stored in a database, they must be transmitted in a format agreed upon by all trading partners. To that end, standards serve to hide as many of the technical storage details (e.g., entity relationships, data types) as possible.

HIERARCHICAL DESIGN

Instances validated against hierarchical schemas will be interoperable so long as the container elements contain entity data in a standard format. The physical arrangement of data elements shields most if not all of the technical implementation details of the source database from trading partners. What is most notable is that the manner in which instances containing entities with unique identifiers accomplish the task. Unique identifiers for each entity are found in data elements nested within them. The identifiers, therefore, are assimilated inconspicuously into the collection of data contained within the instance.

RELATIONAL DESIGN

Instances validated against relational schemas may also utilize data standards and therefore provide a level of abstraction. However, they lend themselves more easily to tight coupling between database tables and instances because the table metadata and key relationships may be replicated exactly in the instance. Relational instances accomplish unique identification through metadata attributes of data elements—particularly the *keyRef* attribute—rather than through nested data

elements. This maximizes the exposure of the identifier because it serves as the basis for the structure of the instance.

ANALYSIS

Because the architectures vary widely among individual states and because architectures vary within the EPA, XML data interoperability is critical to the success of the Exchange Network. Central to this interoperability is logical abstraction between XML schemas and any trading partner's relational database structure. In other words, data relationships in an instance should look as little as possible like the relationships among corresponding tables.

XML instances that conform to hierarchical schemas are better suited to that end. Because the data are denormalized and related instead through physical nesting, a hierarchical structure betrays nothing of the key relationships among the data that may exist in some database. Consumers of the data can then choose to renormalize the data however they wish so that they may be incorporated into the destination database.

It is theoretically true that an XML instance that conforms to a relational schema may not necessarily mirror the key relationships found in the database where the data originated. However, it is a practical reality that the temptation to do so will be too great to ignore because of the difficulty of moving from one relational structure to an alternative relational structure. Database facilities, for example, export XML instances in a relational structure that mirrors the table structure. While such an approach saves time, it compromises interoperability.

It should also be noted that rote replication of the database key relationships in an instance poses a security risk. Betrayal of a database structure by an instance conveys information that has the potential to aid an attacker. This is a case in which loose coupling promotes not only interoperability but stronger security as well.

Message Size

Message size refers to the file size of a typical XML instance exchanged between participants in a business process step. This criterion is especially important since the size of the documents exchanged between two system endpoints will directly affect the performance of the system. File size impacts the amount of bandwidth that is used when transferring messages as well as the time it takes to create and process a document.

In relational database tables, one of the primary advantages of normalization within a database is the reduction of its memory footprint as data repetition is minimized. A similar phenomenon can occur with XML schemas.

HIERARCHICAL DESIGN

XML instances validated against hierarchical schemas are conceptually analogous to denormalized relational database tables. Every relationship between one data element and another is expressed through nesting. For situations in which a single data structure is associated with many others, then the structure will be repeated needlessly and in its entirety wherever it is needed. The result is unnecessarily large XML instances.

RELATIONAL DESIGN

XML instances that conform to relational XML schemas contain normalized data. This normalization is similar to relational database tables so that data duplications (as found in the hierarchical design) are replaced by more efficient relationships between data structures.[1]

Specifically, many-to-many data models will require that the same data structure repeat many times (as it is referenced by other data structures). Also, some one-to-many data models may find the same situation to be true. In each of these cases, a relational design can reduce the verbosity of conforming XML instances by leveraging references to data rather than incurring needless repetition of the same data.

With a relational design, data elements appear once in an instance with unique identifiers affixed to them. Other structures that need to declare relationships with these elements do so by declaring references to those identifiers rather than containing the elements themselves. As a result, the typical message size will be much smaller than its hierarchical counterpart.[2]

ANALYSIS

As would be expected, larger instances result in slower transmission speeds across a network and demand greater resources (e.g., processor speed, memory) to process the messages. Also, XML processing mechanisms such as DOM and SAX parsing and XSLT will perform more slowly against large XML instances. For

[1] In relational table structures, an association table between two entities in a many-to-many relationship has an upper bound of $m \times n$ records (where m is the number of records in the first table and n is the number of records in the second table). Similarly, an XML instance could contain up to $m \times n$ representations of those associations. In addition, each of those associations will contain all the XML elements pertaining to the individual records of the entities. The result is a document containing up to $(m \times n) \times [(m \times f_1) + (n \times f_2)]$ elements, where f_1 is the number of fields contained by the first entity and f_2 is the number of fields contained by the second entity. This calculation assumes only two entities in a many-to-many relationship. The order of multiplicity grows with each additional entity.

[2] XML instances will contain up to $(m \times n) + (m \times f_1) + (n \times f_2)$ element representations for two entities in a many-to-many relationship, fewer than that derived from its hierarchical counterpart.

these reasons, relational schemas typically lead to superior performance because they result in smaller instances. When numerous many-to-many relationships are being modeled, as is the case in the RCRAInfo Compliance Monitoring and Enforcement schema, a hierarchical instance is likely to have an extremely large file size. Therefore, relational schemas are far superior to hierarchical schemas when considering message size and performance.

Readability

The ability for humans to read XML schemas and instances is often touted as a benefit of XML technology. Developers and analysts alike can understand the semantics of a message by reading through the associated XML instance text.

HIERARCHICAL DESIGN

Hierarchical designs are very approachable for business analysts. When presented graphically, they provide a clear overview of how the schema is constructed by depicting the layout of the physical structure of the expected data.

RELATIONAL DESIGN

While relational designs may be intuitive to developers, business analysts may have a hard time understanding this type of design. Even when presented graphically, the physical structure of a relational design approach takes time to understand. References among data elements that serve to reduce data redundancy can obscure the relationships that constitute a message.

ANALYSIS

Hierarchical designs provide a higher level of human readability than relational designs.

Processing Ease

Processing ease refers to the ability for software programs to process XML instances by transforming them to different XML structures, database imports and exports, etc. When deciding whether a relational or hierarchical design is appropriate, the way in which the XML will be processed by the receiving or intermediary parties should be considered carefully. More complex processes are likely to have a greater possibility of error, potentially compromising the integrity of the data in XML instances. For example, a receiving party may simply convert an XML document to data in a software application (e.g., DOM objects) and then into a database. Alternatively, other systems may need to modify an XML instance (simple or otherwise) before sending it to another destination. The extent to which direct modifications are required to XML instances will impact the design most appropriate for those documents.

HIERARCHICAL DESIGN

If outgoing data come from a normalized relational database, then instances validated against hierarchical schemas may be more difficult to produce than relational instances. This is because the data will need to be denormalized to generate an instance. Automated database mechanisms, especially in XML-enabled databases, may ease this transformation. Otherwise, programming code needs to be written to perform the denormalization in such a manner as to preserve data integrity.

From the perspective of a receiving application, the consumption of hierarchical instances can be arduous because of large document size, and the transition from the denormalized instance to a normalized database could be problematic. However, developing XSLT stylesheets to transform an XML document to another format is reasonably straightforward. Defining the mappings between the denormalized XML and the resulting output files should not demand any complex XPath expressions within template definitions.

RELATIONAL DESIGN

Outgoing instances that conform to a relational schema may be produced more easily because the normalized structure likely to be present in the source database can be reproduced in an outgoing XML instance through *key* and *keyRef* mechanisms (assuming the relational database matches the relational design of the schema). In other words, a software application creating XML instances can leverage the database table relationships as part of its instance creation logic.

Relational instances can be easier to consume because of their relatively smaller size, and the transition from the normalized instance to a normalized database can be straightforward. However, writing XSLT stylesheets to transform relational instances to other relational structures may be difficult. Template definitions require some complex XPath expressions using the built-in *key()* function. Therefore, a sophisticated understanding of XPath is required to generate a reliable stylesheet.

ANALYSIS

The ease with which XML instances can be processed within the Exchange Network is difficult to assess because of the variability in the manner in which XML instances are produced by the states. Some states may use database features like stored procedures while others may use programming code, and still others may use a combination. Also, some receiving systems may not modify an XML document, whereas others will need to perform sophisticated modifications directly to the XML content. Although difficult to ascertain due to the variability of trading partner databases, in general, creating a hierarchical XML instance will prove easier for the majority than creating a relational XML instance that differs from a more than likely different relational source.

Established Practices within the XML Schema Development Community

In our research, we found that many standards organizations have not issued explicit guidance about hierarchical or relational design in their XML design rules. However, we analyzed XML schemas developed by various organizations and the general character of that work as related to hierarchical and relational design.

HIERARCHICAL DESIGN

The U.S. Federal Government, OASIS, and UN/CEFACT all have favored a hierarchical design for schemas and corresponding instances. The Naming and Design Rules (NDR) associated with OASIS and UN/CEFACT (originally cooperative parties in the creation of the ebXML initiative) advocate a hierarchical XML schema design. The rules mandate that XML elements (apart from the root element) within an instance are subordinate to other elements. The Department of the Navy's NDR address the *key* and *keyref* mechanisms within the XML Schema specification, but the rules do not deal with the design considerations that would compel a schema author to adopt a relational model.[3]

RELATIONAL DESIGN

The OpenTravel Alliance (OTA)—a nonprofit XML standards organization for the travel industry—has included some relational design concepts in its schemas, although most of OTA's XML schemas favor hierarchical structures. The OTA does not implement its relational approach with *key* and *keyRef* but rather with a proprietary design called a Reference Place Holder (RPH). The RPH achieves the same purpose as *key* and *keyref* by providing a reference to an element by way of an attribute within a separate element. This reference decouples content defined in the schema and supports the creation of relational models. Like the Department of the Navy's NDR, though, the OTA's *XML Schema Design Best Practices* do not address specific design considerations that would compel a schema author to adopt a relational model (via the proprietary RPH mechanism in this case).

ANALYSIS

On the whole, voluntary consensus standards organizations are creating XML schemas according to a hierarchical design. However, as these XML schemas are used increasingly in production systems, with an increasingly large volume of data, issues associated with this design may emerge, requiring the standards-setting bodies to adjust their XML schema design guidelines accordingly. Therefore, we believe that the precedent set by prevalent industry standards organizations is incomplete and does not provide an established example for the Exchange Network to follow.

[3] See http://xml.coverpages.org/DON-XML-NDR20050127-33942.pdf.

CONCLUSION

In conclusion, Table 1 lists the evaluation criteria in order of importance and, for each, indicates whether the hierarchical or relational design better meets the needs of the Exchange Network. The relative importance of each criterion is a function of both the Exchange Network business process in which XML instances are used and the needs of the stakeholders who execute the various phases of the process.

Table 1. Summary of Evaluation

Criterion	Recommended design	Explanation
Data integrity	Relational	A relational design is appropriate for state-to-federal flows because these instances are likely to go through some transformation that would be more error prone with a hierarchical model. Relational XML instances are typically smaller and should reduce the frequency of data integrity failures during processing.
Interoperability and loose coupling	Hierarchical	A hierarchical design provides a layer of abstraction and thus loose coupling of data to the typical relational database source.
Message size	Relational	A relational schema promotes smaller message sizes, especially when many-to many data relationships are involved.
Readability	Hierarchical	A hierarchical schema is often easier for developers and business analysts to read and interpret.
Processing ease	Hierarchical	A hierarchical instance provides a more straightforward mapping to all trading partners. Trading partners with relational sources would be required to map to a different relational instance.
Established practices within the XML schema development community	Inconclusive	Prevalent standards bodies have adopted a hierarchical model, but no extensive evidence suggests that this was deliberate according to well-defined criteria.

Our analysis of hierarchical and relational XML schema designs suggests that neither approach is overwhelmingly more suitable than the other for use by the Exchange Network. For some situations, a hierarchical design is appropriate, while for others, a relational design is most efficient. The right design will depend on the specific requirements for each information flow, as related to processing, message size, etc.

A hierarchical schema design is best for one-to-many data models in which container elements associate an entity with multiple instances of another entity. Benefits include a greater propensity for loose coupling, a greater level of readability and in, general, greater processing ease. Typically, XML data exchanges are comprised of data structures with relatively simple associations to other data structures. In those cases, the Exchange Network should make use of hierarchical XML schema design methods.

In contrast, many-to-many data models are best conveyed through a relational design. Further XML data exchanges that contain data that recur within many other

data elements are also best suited for relational designs. Benefits include smaller message size and better data integrity. In those cases, the Exchange Network should make use of relational XML schema design methods.

The Exchange Network should establish guidelines that will help schema developers decide whether to use relational data structures with references or traditional hierarchical structures. This guidance should extend directly from the analysis criteria outlined in this position paper. The two designs can, and should, coexist within the same XML schema and conforming instances.

The Exchange Network also will need to define the mechanism (e.g., procedural steps) by which Core Reference Model (CRM) entities can be consistently translated into XML schemas. By establishing these guidelines and leveraging the benefits of both hierarchical and relational schema models, the Exchange Network will create an environment conducive to a consistent modeling approach as well as XML instances that are as efficient as possible. Ideally, if the design approach is defined objectively, creating XML schemas can be automated when based on the CRM.

NOTICE:

THE VIEWS, OPINIONS, AND FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF LMI AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL AGENCY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER OFFICIAL DOCUMENTATION.