Structure and Design Issues for Developing, Implementing, and Maintaining a Justice XML Data Dictionary

Report of the
Justice XML Structure Task Force

Office of Justice Programs,
U.S. Department of Justice

August 23, 2002

Acknowledgements

The authors and the Justice XML Structure Task Force wish to acknowledge a variety of contributions (inputs, feedback, discussions, etc.) by individuals from the following alphabetical list of agencies, organizations, institutions, and companies:

American Association of Motor Vehicle Administrators (AAMVA)
Bureau of Justice Assistance, Office of Justice Programs, U.S. Department of Justice
Criminal Information Sharing Alliance (CISA)
Executive Office of the U.S. Attorneys
Federal Bureau of Investigation
Georgia Bureau of Investigation
Global Justice Information Network, Infrastructure/Standards Working Group (ISWG)
Greacen Associates LLC
Illinois State Police
Institute for Intergovernmental Research
Integrated Justice Working Group, LegalXML
Joint Task Force on Rap Sheet Standardization
Los Angeles County Sheriff's Department (California)
Minnesota Bureau of Criminal Apprehension
Missouri Supreme Court
Mobiam Solutions, Inc.
National Center for State Courts
National Institute of Justice
National Telecommunications and Information Administration (NTIA) – Technical Lead
New Mexico Department of Corrections
Oklahoma Department of Public Safety
Practitioner Resource Group
Regional Information Sharing Systems (RISS)
SEARCH, The National Consortium for Justice Information and Statistics
Sierra Systems
Southwest Alabama Integrated Criminal Justice System
SouthWest Border States Anti-Drug Information System (SWBSADIS)
Templar Corporation
Unisys Corporation
Wisconsin Department of Justice

Structure and Design Issues for Developing, Implementing, and Maintaining a Justice XML Data Dictionary[1]

Executive Summary

This report addresses structure and design in implementing a Justice XML Data Dictionary that facilitates effective information exchange across a diverse professional community. It documents a starting point for, and marks the progress of the Justice XML Structure Task Force (XSTF). It does not necessarily address all issues, nor does it present the only solutions. It addresses major issues of requirements, design, structure, and implementation considered by the XSTF.

This report is intended to:

1. Expose some of the important structural and design issues associated with implementing a standard XML data dictionary, an associated XML schema, and a potential registry/repository.

2. Identify some potential solutions and best practices from collective experiences encountered in other projects.

3. Identify and discuss the advantages and disadvantages of potential solutions.

4. Identify critical decision points for the Global Infrastructure/Standards Working Group (ISWG) XML Committee.

5. Recommend courses of action for these decision points.

---

[1] This report was prepared by Mark Kindl and John Wandelt, Georgia Tech Research Institute (GTRI), on behalf of the Justice XML Structure Task Force. It is a product of the Office of Justice Programs (OJP), United States Department of Justice, and supports OJP integrated justice information sharing initiatives. The Justice XML Structure Task Force (XSTF) is a component of the Justice XML Subcommittee, which is a part of the Global Justice Information Network Advisory Committee (GAC) Infrastructure/Standards Working Group. The GAC was created to promote broad-scale sharing of critical justice information by serving as an advisory body to the Assistant Attorney General, OJP, and the United States Attorney General. (For further information on the GAC initiative, visit http://it.ojp.gov/global.) The XSTF was created to build on the work of the Justice XML Subcommittee Reconciliation Data Dictionary (RDD). Research for this report was provided by GTRI (http://www.gtri.gatech.edu). For other background documents, please visit http://www.search.org/xml/default.asp, the Web site of SEARCH, The National Consortium for Justice Information and Statistics.

Special Note: This is a *living/working* document that the XML Structure Task Force will update periodically as the Justice XML Data Dictionary Schema develops. It is intended to examine and review lessons learned and design decisions.

6. Identify design and structural issues that may require further attention: research, experimentation, evaluation, discussion, review, and/or decision.

What is meant by *structure* in the context of standards for information exchange? Fundamentally, effective information exchange requires common syntax and semantics. Syntax defines the standard rules for the arrangement of the data and information items in exchange instances. The Extensible Markup Language (XML) establishes a formal standard syntax specification. Semantics are usually recorded in a data dictionary (DD) and establish the standard meaning of the data and metadata to be exchanged. Nonetheless, even with standard XML syntax and well-defined semantics, effective communication and information exchange require structure; that is, an information model and rules that constrain syntax to ensure semantics are represented consistently and unambiguously.

Background

Collectively, rapidly growing caseloads, diminishing resources to process cases, and the increased mobility of criminals and society have combined to increase exponentially the need for complete, accurate, and timely justice system information. Justice system officials realized the need for immediate access to more information and the means to move it electronically between organizations. More importantly, they recognized the necessity of a comprehensive, strategic, enterprise-wide approach to information sharing and integration between agencies.

The establishment of a Justice XML (Extensible Markup Language) Data Dictionary is one part of a strategy designed to enable effective information sharing and communications among the many justice/public-safety agencies at the local, state, and federal levels. The development and maintenance of the Justice XML Data Dictionary is sponsored by the U.S. Department of Justice, Office of Justice Programs (OJP), which works closely with the representatives of the Global Justice Information Network (referred to as Global). Global is a Federal Advisory Committee that advises OJP, and recommends courses of action related to the information sharing needs of the justice and public safety communities.

The initial development of the Justice XML Data Dictionary was accomplished in conjunction with the analysis and reconciliation of three different XML specifications. The *Interstate Criminal History Transmission Specification* [1], the *Regional Information Sharing Systems (RISS) Data Exchange Specification* [2], and the *Electronic Court Filing Standard* [3] were reconciled, despite the fact that they satisfied functional requirements associated with independent business plans. Principles and procedures were established and agreed to by all participants that preserved the ability of all involved to effectively perform their prescribed operations. The initial product of this group was known as the Reconciliation Data Dictionary (RDD) [5]. Later, the American Association of Motor

Vehicle Administrators (AAMVA) joined the group and brought new data elements into the RDD.

To date, the RDD Task Force, the XML Committee of Global Infrastructure and Standards Working Group (ISWG), and the National Telecommunications and Information Administration (NTIA) have established a baseline for the Justice XML Data Dictionary (DD) and its associated principles and concept of operations. There are many more design and structural issues to resolve regarding a corresponding information model and XML schema implementation. These should remain independent of specific databases or other application systems if they will effectively serve the justice and public safety community at large. They should define, standardize, and facilitate how the XML components and structures are designed, (re)used, extended, and maintained.

In December 2001, Global ISWG XML Subcommittee identified two major categories of issues for future action:

- Governance: the establishment of the policies, procedures, and rules of order for administering, maintaining, supporting, and promoting the dictionary standard.
- Structure: the establishment of a standard technical design and structure for an XML schema implementation of the Justice XML DD.

In May 2002, the XML Subcommittee of Global ISWG met in Charleston, South Carolina, and commissioned the XML Structure Task Force (XSTF), under the chairmanship of Mr. Paul Embley, to review technical design and structural issues, and to recommend how to move the Justice XML DD to a standard Justice XML DD schema implementation.

This extract from notes of that XML Subcommittee meeting sets a course for the XSTF:

> *Everyone wants a dictionary that is itself a schema—which developers can point for elements and objects. The consensus was that this will make the dictionary much more valuable, and the implementations of it much more precise. Consequently, it is worth taking some more time to get from here to there before "locking down" the dictionary. This will require ... a better approach than an endless series of reconciliations as more and more groups come to the table. The "Embley" committee* [XSTF] *will try to come up with an approach that keeps as many as possible of the agreed-upon elements intact and makes sure that persons who implement using the current dictionary are not forced to move to the new dictionary at any specific time.*

The XSTF brings together a diverse group of domain experts, users, and developers with a variety of experience. Participants represent developers, users, and domain experts from at least four different ongoing information exchange projects, as well as state and federal governments, industry, academia, and user groups.

The XSTF met several times both face to face and via telephone conference during May, June, and July 2002. During this series of meetings, the XSTF identified, discussed, and formed recommendations for the resolution of technical issues related to developing a standard Justice XML Data Dictionary Schema (JXDDS) associated with the Justice XML Data Dictionary (DD). A plan for moving from the current RDD to current and future standard XML schema implementations was developed and is presented herein.

Recently, Global ISWG has made it a priority to ensure that these developing justice data standards are consistent with other similar government efforts and will adhere to federal guidance. To this end, Global is reaching out to coordinate with and learn from other XML efforts, most recently XML.gov.

Assumptions

The XSTF derived the following assumptions from the participants' collective experiences and expertise with data systems development and the justice domain. In many cases, these assumptions are requirements for the Justice XML DD and schema and will be addressed in more detail within this report. The following are assumed:

- The Justice XML DD should:
  - Maximize information sharing.
  - Be reusable, extensible, and maintainable.
  - Employ current information systems technologies and best practices.

- Adoption of a Justice XML DD will be voluntary and widespread.

- The exchange of information should go beyond simple sharing via static bilateral agreements and, instead, should evolve to become universally standard throughout the justice community.

- An effective standard XML-based exchange environment requires the design and synthesis of a common repository of standard, referenceable, reusable, extensible data components that implement the corresponding semantics and structure of the specifications in the Justice XML DD.

- The Justice XML DD and DD schema should be based upon a conceptual information model that is generalized for the justice community and does not target specific applications, databases, or business documents.

- Components should primarily target schema developers by providing components to build XML schemas for both (1) documents (persistent records created for business purposes), and (2) data transactions (possibly transient data/info for query and response).

- The Justice XML DD and its associated XML schema will evolve and require modification and extension over time. The architectural design of the dictionary and schema should facilitate relatively easy maintenance and modification.

- The best design for extension and maintenance will minimize impact on prior investments in schemas, applications code, and integration.

- The static exchange environment will evolve and become highly dynamic and information rich. In addition to sharing, there will be a need to share relationships and knowledge, and mine, discover, or create new information and knowledge.

- There is a requirement to represent and implement domain relationships between information objects so they are understood and recognized globally.

- There are no perfect solutions and no "silver bullets." Technical dependencies in requirements and solutions, as well as time constraints resulting from national priorities and demands, will force the rational consideration of many trade-offs.

Technical Issues and Findings

The *Issues and Findings* of this report are generally organized in a sequence that progresses from early concept and design to implementation.

Initially, we identify the need to establish formal written requirements and to supplement the Justice XML DD with a formal schema specification. Then, using object-oriented data concepts, we suggest one possible framework for designing the information objects, and their relationships. This framework results in a dictionary that is easy to maintain and extend, and that makes maximal re-use of the objects and their component parts. We provide examples of how to implement abstract reusable data objects (types) in an inheritance hierarchy, how to extend them, and rules for maintaining their properties in XML schema. We also provide examples of deriving new elements (tag names) from the types, and the need for and examples of primitive and other support types.

We present a possible architecture for organizing and referencing the schema parts and associated namespaces from XML instance documents. This establishes a logical and standard structuring to the XML exchange environment.

Suggestions are presented for issues relating to data constraints, metadata, versioning and compatibility, and business containers and components.

Component sizing, structure, and the representation of relationships between them are addressed in some depth. Sizing is important because it significantly impacts component

reusability and flexibility. Relationships are important because they provide the context that turns data into information and, ultimately, information into knowledge on a global scale. We also briefly discuss other related standards, new emerging Internet technologies, and the potential for using them.

The *Issues and Findings* of this report are organized around major issues. Each issue is introduced as a question or set of related questions (the same format the XSTF used). Each issue is addressed with a summary discussion about its importance and impact. Each issue has a recommendation that summarizes the XSTF collective opinion. As appropriate, this may be followed by a summary of advantages, disadvantages, and one or more examples that illustrate aspects of the issue or potential solutions. Sometimes additional questions are raised as a result of discussions, and we add these where applicable. Finally, some issues are not resolved and may require additional discussion, research, or advancements in technology. We have tried to flag such issues with appropriate suggestions for necessary actions.

Plan and Recommendations

We conclude this report with a plan and recommendations for evolving the Justice Standard XML Data Dictionary from its RDD v1.0.0 roots to a robust, formally defined and structured XML schema that is easily used (and reused), extended, and maintained.

In summary, the XSTF recommends a number of actions:

1. Use the most recent RDD version 1.0.0 to generate Justice XML Data Dictionary Schema Specification v2.0.0 for immediate use by organizations that have already begun to implement applications with RDD v1.0.0.

2. Use RDD v1.0.0 to generate Justice XML DD Schema Specification v2.1.0 that employs upper camel case for XML schema types and elements (lower camel case for attributes).

3. Develop and deploy a translation mechanism to facilitate compatibility between Justice XML DD Schema (JXDDS) versions 2.0.0 and 2.1.0.

4. Begin to design and implement JXDDS v3.0.0 that employs object-oriented data design principles and a schema reference architecture as detailed in this report. This implementation will be more easily extendible and will incorporate the capability to represent domain relationships through a limited form of the World Wide Web Consortium (W3C) Resource Description Framework (RDF) specification. This version should also incorporate data element naming guidelines of ISO/IEC Standard 11179 *Specification and Standardization of Data Elements* [5].

5. Prototype a translation mechanism to generate JXDDS v3.0.0 from JXDDS v2.1.0 as a migration path to the advanced version.

6. As W3C RDF technology matures with respect to XML schema, begin to design and implement JXDDS v4.0.0 that employs a full RDF capability in XML. This can potentially enable autonomous reasoning with and about information objects, relationships between them, and patterns that emerge.

7. Prototype a translation mechanism to generate JXDDS v4.0.0 from JXDDS v3.0.0 to facilitate migration to advanced RDF-based version.

The first three actions can be executed almost immediately. Working prototypes have already been developed and can be viewed at:

http://justicexml.gtri.gatech.edu/jxdds/XML_Schemas.html
login: justicexml / password: justicegtri

or

http://justicexml:justicegtri@justicexml.gtri.gatech.edu/jxdds/XML_Schemas.html

Actions 4 and 5 are estimated to require from 6 to 12 months of effort. Actions 6 and 7 are dependent upon W3C RDF specifications currently being worked, and the development of tools associated with those specifications.

This action plan was developed to capitalize on the current excitement and potential of XML technology and a need to respond quickly to national priorities. It attempts to leverage and balance the trade-offs among time constraints, investments in recent standards work, the capabilities of today's technology, and the requirements of a large, distributed, heterogeneous, data-rich operational environment.

Table of Contents

Section **Page**

(This page is intentionally blank)

Issues and Findings

1. Standard XML Data Dictionary Schema

Should an XML schema be implemented for the Justice XML Data Dictionary (DD)?

Discussion / Importance / Impact

An XML schema is an unambiguous specification for the representation of reusable information components expressed in XML. The current Microsoft Access (and PDF) distribution of the Justice XML DD is non-normative (because it has multiple XML representations), is not referenceable (because there is no standard schema of defined components), and cannot be validated in its current form (because it exists informally in small non-normative example parts). Distribution of a standard dictionary in the form of an XML schema is consistent with the W3C (World Wide Web Consortium) architecture and facilitates component reuse and standardization. An XML schema should be derived from a standard justice and public safety data model (information objects, their composition, and their relationships). The underlying data model should not limit the utility of the Justice XML Data Dictionary in the broader community.

The XSTF recommends that a standard XML schema should be developed that implements the Justice XML DD.

Pros

Implementing a standard Justice XML DD schema provides:

- A referenceable specification.
- Normative (standard) definitions of components.
- Unambiguous definitions of components with ability to validate using XML tools.
- Re-usable (and therefore efficient) components.
- A basis for namespace control.

Cons

Requires central maintenance and some level of user support.

Example

A prototype of a Justice XML Data Dictionary schema built from the Reconciliation Data Dictionary v2.0.0 can be examined at:

http://justicexml.gtri.gatech.edu/jxdds/XML_Schemas.html
login: justicexml / password: justicegtri

or

http://justicexml:justicegtri@justicexml.gtri.gatech.edu/jxdds/XML_Schemas.html

2. Requirements Identification and Definition

What are the requirements for the Justice XML DD and associated DD schema? What *use cases* describe how the Justice XML DD and DD schema will be employed?

Discussion / Importance / Impact

Requirements identification and definition are probably the most important aspects to designing any information system specification (standard or otherwise). Failure to identify, define, and document requirements early traditionally results in the most costly mistakes and rework.

Important question: Who are the *real users* of the Justice XML Data Dictionary? If the only purpose of the dictionary is to standardize exchange format, then court administrators, law enforcement officers, etc., who use end-to-end applications that exchange information in XML should never see the content of the dictionary. This implies that the *real users* will generally be technical people who design, develop, and maintain justice systems, software, and schemas. We add this to our list of assumptions.

Users and technicians should jointly develop, document in writing, and agree to a clear, unambiguous statement of requirements for the Justice Standard XML Data Dictionary and associated specifications. The statement of requirements should identify and define the functional capabilities that this XML specification will provide or enable. It should detail the functional capabilities that users need, desire, and expect, and it should enable standard developers to better understand how to provide these capabilities.

A written requirements definition is always the foundation for testing, validation, and verification. Without some written record of required needs and expected capabilities, it will not be possible to design metrics to ensure that a specification can perform as desired. Furthermore, there will be no way to determine the completeness of a work-in-progress specification. The Justice XML DD currently has no written requirements. Without understanding specific uses of the Justice XML DD, it is difficult to determine the best reference architecture and overall design.

Use case analysis and definition is an easy method for documenting requirements. The basic idea is to create example scenarios that illustrate typical uses of a product *(use cases)*. Domain experts develop a set of written scenarios that describe how the Data Dictionary will be used and how it is expected to operate from the user perspective. As appropriate, use case scenarios address different kinds of users (e.g., system developers, schema designers, maintainers, etc.) and typical examples of usage. One scenario may cover a single or multiple services or requirements. The objective is to create a set of use cases that cover most (if not all) of the critical requirements and functional capabilities that implementers would otherwise have to research, assume, or guess.

Recommendations

Some form of written requirements definition is necessary for the Justice XML DD, JXDDS, a schema reference architecture, registry/repository, and the overall information exchange environment and its intended uses. Some assumed requirements for a reference architecture are listed below in the *Example* section (based on group experience and discussion). A committee of domain experts, user representatives, and developers should confirm these requirements and then recommend that use case scenarios be documented in writing to detail the most important functional capabilities expected by the justice community (i.e., from a user perspective).

Pros

Defining and documenting requirements ensures:

- Common understanding between using community and developers.
- That user knows the capabilities that he needs, wants, and expects.
- That developer understands user expectations for functionality.
- That specification will meet identified justice expectation of capabilities.
- A basis for user interfaces, tools, and other needs.
- Criteria for testing, validation, verification, and completeness.

Use case analysis is an easy way to document requirements in plain English.

Use cases do not have to be 100% complete to provide utility to designers and/or developers.

Use case documentation can be work-in-progress.

Cons

- Requirements identification, analysis, and definition are almost always difficult.
- Can be a lengthy process of controversy, compromise, and, hopefully, consensus.
- Usually change with time, technology, and needs.

Example

Example requirements for the schema reference architecture may include:

1. The reference architecture must be extensible.

2. The reference architecture must support the capability to merge and resolve the content of similar documents (e.g., multiple Rap Sheets for the same criminal).

Such requirements can usually be more succinctly described and understood through use cases. Although use cases have not been written for the Justice XML DD, we present a simple example of one possible *use case* for the JXDDS. This particular use case is not only an example, but may also serve as a key requirement for the registry/repository concept envisioned for the Justice Data Standards. Note that this use case reveals many requirements, including one class of user, registry capabilities, registry organization, XML schema editor, and others. Also note that while many requirements are explicitly stated, there are many others that may only be implied. It is not necessary to ensure that every requirement is covered explicitly. Developers can always ask questions about implied requirements; use cases give them something to base their questions on.

Use Case: A technical user must design and implement an XML schema.

Given: A state employee assigned to the Information Systems Division (i.e., a technical user) in the Department of Corrections is required to design and implement an XML schema to transmit a standard semiannual report listing the state prison inmates who are scheduled for release within the next 180 days. The employee is familiar with the data required by the report and the database that the data will come from. He is also familiar with XML schema. He is not familiar with all the items in the online Justice XML Data Registry (a very large registry of well over 2000 element names and types). However, he knows that he can either use a whole XML schema specification (extending it, if necessary), or build his own schema by importing the Justice namespace and reusing the standard elements and types therein (extending them, if necessary).

Scenario: The state employee reviews his data requirements and then accesses the online Justice XML Data Registry. Using an object-oriented view of the complex types, he browses the registry, locating and confirming all the standard type definitions for the data he will need to assemble the report. These include an *Inmate_Type,* a *CriminalJusticePerson_Type,* a *Parole_Type,* and an *Address_Type.* To locate these type definitions, he browses a catalog in the form of a type hierarchy that is organized from general to specific. For example, to find *Inmate_Type* and *CriminalJusticePerson_Type,* he browses from the root type to *Person_Type,* and then finds that *CriminalJusticePerson_Type* and *Inmate_Type* are derived from *Person_Type*. He also checks all the standard element names that use those types to determine if he can use standard elements or if he must create his own from the standard types. The registry enables him to look up the definitions and other metadata associated with each element and type so he can confirm that these meet the requirements of his report schema.

Since the report will be in the form of a list of persons and associated data, he also wants to find some form of standard *List_Type*. He uses the registry search capability to determine if any such type exists. It does not, therefore he will create his own from standard types.

Once he has identified all the appropriate standard types and elements from the registry, he uses an online XML schema editor to build and validate his schema from standard types and elements. (End of use case example.)

Related Questions

For what purpose must you exchange data? How often? With whom? How much data?

What is the protocol for data exchange; for example, query/response, standing query, subscription, remote procedure call, or all of these?

Will data exchanges accumulate data? Accumulate document records?

Are interfaces to other systems needed; for example, interface to electronic records management systems or remote search capabilities?

Are there different classes of users?

Who implements ArrestWarrant.xsd, ArrestWarrant.xml, ArrestWarrant.xsl?

What data entry or editing tools are needed? Other kinds of tools?

How do you know if all requirements have been identified? Clearly defined? Satisfied?

Are all capabilities described in the use cases implemented and operational as described?

---

3. Conceptual Data Modeling Framework

Is it necessary to derive the data dictionary and associated XML schema on the basis of a conceptual data model? If so, what kind of data model?

Is it necessary to capture and define relationships between information objects?

What entities and relationships exist and should be captured for the Justice XML DD?

Discussion / Importance / Impact

A data dictionary defines the metadata labels for the entities and properties (attributes) of a database or set of databases. It provides common understanding for the terms (metadata) used to describe data values.

A data dictionary can simply be a written document of metadata definitions used as a human reference (such as Webster's Dictionary). It can also be a catalog of metadata rules (essentially machine-readable definitions) built for and used by database management system software to understand how to manipulate data.

XML provides the capability to do both in parallel. This is both good and bad. XML provides standard structure (syntax) for automated data exchange and processing. At the same time, humans are able to read, define, and structure the data for human understanding. Without a well-planned design built on a model of the data, XML can circumvent database principles and result in schema designs that do not encourage reuse

and are difficult to extend and maintain, especially for a large diverse community of users.

A logical data model is a collection of concepts that describe data, its properties, relationships, and constraints. It is a common concept for objects, their structure, and the relationships between them. Fundamentally, there are three kinds of relationships in a data model: *IS-A* (or *TYPE-OF* or *IS-TYPE*), *HAS-A (*or *PART-OF* or *HAS-PART*), and domain relationships (e.g., person *OWNS* vehicle or weapon *USED-IN* crime). In constructing objects, it is considered a best practice to focus on the *IS-A* and *HAS-A* relationships first, especially since time is limited, and this is consistent with the existing Reconciliation Data Dictionary approach. The JXDDS should implement entities that represent common objects in the justice and public safety domain (as captured by the Justice XML DD). Furthermore, standard business container components (for introductory headers, digital signatures, security notations, privacy notations, etc.) should also be included in the dictionary and implemented in the JXDDS.

To establish a robust standard XML information exchange environment that is reusable, extensible, and maintainable, defining a set of tag names and structures is necessary but not sufficient. There should also exist a conceptual model that provides some formal framework for the types, elements, and attributes that are defined. Ideally, this model should provide an extension mechanism that facilitates local adaptation without requiring major changes to the baseline standard.

Establishing standard XML tags will enable data exchange. However, the ability to exchange data does not immediately imply that the receiver (machine or human) unambiguously understands the data, how to use it, how to process it, what data is available, or what data should be sought. This requires knowledge of data constraints, relationships between objects, and relationships between properties (the kind of knowledge that a conceptual data model provides).

An object-oriented data model maps the relationships of general concepts to specific concepts and real things. Such a model can be the basis for an XML schema design that is easy to:

- Use and understand, because it models how we classify and use real world concepts and objects.
- Extend and maintain, because baseline types and rules for their use and modification are formally defined and documented.

It is also important to represent domain relationships. However, XML schema does not represent domain relationships well, and there is no accepted standard method to do this in XML. The emerging standard layers of the *Semantic Web* (RDF: Resource Description Framework and its extension OWL: Web Ontology Language) are addressing these issues for the Internet. These technologies can represent domain relationships more effectively and they still use XML syntax. We expect that future software products will be "aware" of and use this technology to understand and act more intelligently.

However, in the interim, domain relationships can be captured within the standard documents (Rap Sheet, Arrest Warrant, Incident Report, etc.) as they currently are. The XSTF believes it will be useful to identify, catalog, and develop a mechanism to represent domain relationships globally as soon as possible.

Recommendations

Some form of data modeling is always necessary for any large, data-intensive project. There should be a data model associated with the Justice XML DD and DD schema that addresses the entities required together with their structural relationships (*IS-A* and *HAS-A*). Domain relationships can be handled within the standard documents that are being and will continue to be constructed from the standard Justice XML DD schema components. However, standard mechanisms for representing domain relationships globally should be addressed as soon as is practical using new technologies emerging from W3C standards efforts.

Since the current RDD is object-based, there is already an implied predisposition for an object-oriented schema design. A recommendation should be made to adopt formal object-oriented data modeling techniques to standardize an XML schema implementation. This will provide better semantic meaning to the relationships between data and information objects.

Pros

The Reconciliation Data Dictionary is already object-based.

A conceptual data model:

- Provides an efficient, abstract, and formal structure for organizing types, attributes, and elements.
- Provides a conceptual blueprint for the information domain against which XML schema objects can be implemented and mapped consistently.
- Facilitates maximal reuse (of objects; i.e., types); minimizes redundancy.
- Can expose potential areas of the domain for which types and elements may be required but have not been defined.

An object-oriented data model:

- Provides a flexible, intuitive, formal extension mechanism.
- Consistently employs well-understood structural relationships (*IS-A* and *HAS-A*).
- Enables better semantic definition of elements (tag names) by logically separating structural semantics (in types) from specific semantics (in elements).
- Organizes objects by their common properties.
- Allows software to treat objects with common properties in a uniform way.
- Allows object types to share a common definition (removes duplicate definitions).
- Can reduce the complexity of designing and using large document schemas by distributing structural semantics into type definitions and associated hierarchy.
- Produces a side effect that any element will be successfully parsed and recognized in any instance that expects one of its ancestors (parents, grandparents, etc.). This is a direct result of maintaining the *IS-A* relationship (described further below).

Cons

Object-oriented modeling is not always simple; real objects (modeled as static) interact, relate, and change through the complexities of processes, actions, activities, and events.

XML schema cannot implement everything (i.e., it is not a "silver bullet"), and there will be some concepts that are difficult or impossible to model.

XML schema has no good way to implement relationships (and knowledge).

Developing the right conceptual data model and framework for the entire justice and public safety community will be very challenging and will require both technical and domain experts.

Examples

Here we provide examples of an object-oriented data model implemented in XML schema. For the following examples, refer to Figure 1 below.

A data type may inherit attributes and elements from a base type from which it is derived. We represent this inheritance in XML schemas by using the extension mechanism. For example, if we wish to create a type *PersonType* to be the base type of *MissingPersonType*, we may create that type and make *MissingPersonType* a derived type. First, we create *PersonType*:

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string"/>
    <xsd:element name="BirthDate" type="xsd:date"/>
  </xsd:sequence>
</xsd:complexType>
```

Next, we create *MissingPersonType* as a type derived from *PersonType*, and add the *MissingDate* element:

```
<xsd:complexType name="missingPersonType">
  <xsd:complexContent>
    <xsd:extension base="personType">
      <xsd:sequence>
        <xsd:element name="missingDate" type="xsd:date"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Note that *MissingPersonType* was created to have three elements: *Name, BirthDate* (inherited from *PersonType*), and *MissingDate*. In object-oriented terms, we say that *MissingPersonType IS_A PersonType* and, therefore, *inherits* all of the properties (elements) of *PersonType* (and will also inherit all properties of the ancestors of *PersonType*).

One effect of the extensibility mechanism in XML schema is that an instance of a derived type may be included where an instance of the base type is expected. For example, in a schema, we define an element of type *PersonType*:

```
<xsd:element name="Person" type="PersonType"/>
```

And then create an XML document that contains an instance of *PersonType*:

```
<Person>
   <Name>Bob</Name>
   <BirthDate>1993-01-01</BirthDate>
</Person>
```

We could create an XML document that contains an instance of *MissingPersonType*, even though an instance of *Person_Type* is expected:

```
<Person xsi:type="MissingPersonType">
   <Name>Bob</Name>
   <BirthDate>1993-01-01</BirthDate>
   <MissingDate>2000-05-18</MissingDate>
</Person>
```

The XML schema validation mechanism may then validate the type of the *MissingPersonType* instance, and ensure that the type instances are consistent with those specified in the schemas.

In order to ensure that the inheritance hierarchy maintains consistency, we insist that a subclass maintain an *IS-A* relationship with its base class. This means if we have a class A, and derive class B from it, an instance of B must also be an instance of A (i.e., B *IS-A* A). We ensure that this is the case by enforcing certain rules on subclasses types.

The following rules for deriving types are imposed on XML schema to ensure the integrity and consistency of the inheritance hierarchy:

1. A derived type may add additional fields (elements/attributes) to its base type.

2. A derived type may restrict one or more fields of its base type, but only so that a derived field is a subset of the field of the base type. For example, a derived type may:
   - Restrict an enumeration from a large set of options to a smaller set of options, as long as every option in the derived set appears in the base set.
   - Remove a field of the base type, but only if that field is optional in the base type.
   - Require a field to appear, but only if the field is optional or required to appear in the base type.

3. A derived type may not modify a field of its base type such that it violates the constraints of its base type. For example, a derived type may NOT:
   - Add additional enumerations to a field.
   - Remove a field that is required by its base type.
   - Modify the type of a field of its base type.

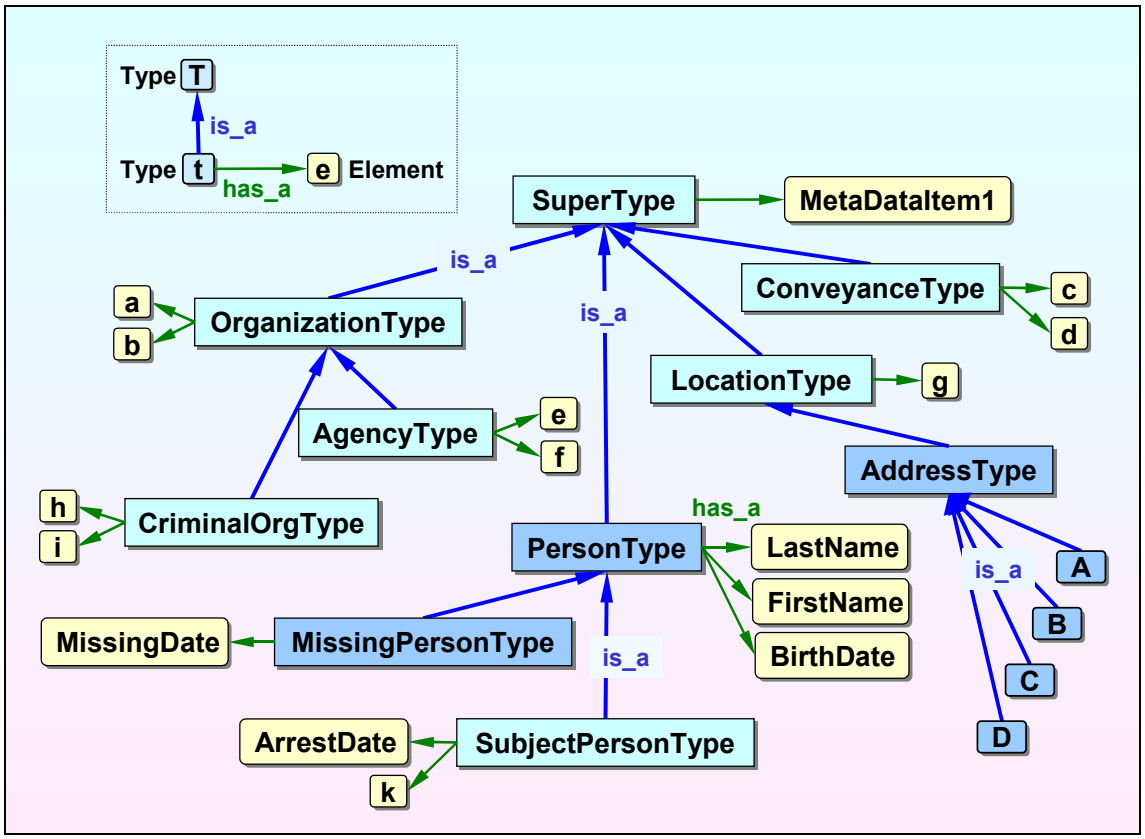

Figure 1. Object Model Example

---

4. Approach to Object-Based Schema Design

Assuming an object-based design, what approach should be used to design the JXDDS?

Discussion / Importance / Impact

There are fundamentally two approaches to object-based designs using XML schema: *subclassing* and *composition*. Subclassing uses the XML *extension* construct to build an object-oriented class hierarchy that enables property inheritance. This design was described in the Examples of Section 3 above.

The other approach to object-based design is *composition*. Composition is simply an *ad hoc* combining of types and elements through nesting and containment. Essentially, this is an exclusive use of the *HAS-A* relationship with a very relaxed meaning; that is, any component (type or element) can be part of any other per user desires and needs. This is the approach used in construction of the current Justice XML DD.

It is also possible to use a hybrid approach that makes appropriate use of both subclassing and composition.

XSTF discussions focused on an object-oriented approach, but everyone's concept of an object was not necessarily consistent. For the purpose of discussion, we refer to *components* as:

- Standard named types, which define data structure.
- Standard element names (defined by standard type names) that carry specific, globally understood semantic meaning.

There was much discussion on technical aspects of building components based upon both subclassing and composition. In summary, the XSTF concluded that discretionary use of subclassing (*IS-A* relationships and *HAS-A* properties) is a useful technique.

Recommendations

The XSTF recommends that subclassing be used as the primary modeling mechanism, but that unnecessarily deep subclassing (hierarchies) should be avoided if possible. A shallow, broad hierarchy of many components that extend from and, therefore, inherit properties (metadata components) from a single root type is desirable. This provides a convenient way to consistently propagate a common set of metadata (properties) to all components.

Subclassing also provides a formal, controlled method for extending the JXDDS. If the baseline hierarchical model can eventually be stabilized, then this will have two major positive implications. First, further evolution of the standard JXDDS by extending the hierarchy at its leaves will have no impact on the baseline already established (except to add new specialized components). Second, local extensions can be created from the standard and used locally without impact to the standard.

Pros

Subclassing:

- Single inheritance (through the propagation of *HAS-A* properties).
- Substitution property within instances (because of *IS-A*).
- Understanding through *IS-A* context provides limited inferencing capability.
- Extension is well defined and controlled. This helps to control compatibility and change. With subclassing, one can:
    - Specialize/generalize with *IS-A* hierarchy.
    - Aggregate with *HAS-A* properties.
    - Add new object entity types (classes) into hierarchy.
    - Add new support types.

    - Insert new properties into existing types.
    - Define element names; add specific semantics through names.

- Complexity is hidden; to understand sub-type, must understand super-type.
- May make it easier to integrate with Resource Description Framework (RDF), a way to represent relationships and knowledge about objects (to be described in more detail later).

Composition:

- Multiple inheritance of properties (through repetition not extension).
- Simple; components are stand-alone (not derived from others); results in loosely coupled design.

Cons

Subclassing:

- Yields a tightly coupled model of components dependent on a root object; and, therefore, is difficult to modify internally within the hierarchy once established. (Note: Although not discussed here, there is an XML schema technique that avoids part of this problem.)
- Consensus on classes and organization of hierarchy is sometimes difficult to achieve.

Composition:

- No substitution property for instances.
- No *IS-A* context, therefore, no inferencing capability.
- Model extension is *ad hoc;* compose new parts from scratch and/or compositions; this makes it difficult to control compatibility even after small changes.

Example

The current RDD is an example of composition. An example of object-oriented XML schema design is outlined in Example subsection of Section 3 above.

---

5. Schema Reference Architecture and Namespaces

What are the requirements for a schema reference architecture? What is the best architecture to address those requirements? What is the best logical organization of the

parts (XML schemas and instances that represent documents and transactions)? How can namespaces be used most effectively? Should a justice target namespace be named or anonymous? Should a justice target namespace be one schema or several logical schema parts? If parts, should any of them be anonymous?

Discussion / Importance / Impact

Based on our earlier discussion, we assume that a standard Justice XML DD schema will be designed for the Justice XML DD and will be widely adopted and used by the community. For a potentially large community of users with a variety of requirements, there will likely be multiple schemas and instances. Therefore, it will be important to consider how these schemas and instances will work together. It will also be important to decide how the schemas and instances should be logically organized for the data exchange environment. We will refer to this logical organization of schema parts and the rules to coordinate them as a *schema reference architecture*.

XML schema provides several mechanisms for combining and coordinating schemas:

- *xmlns:* to reference namespaces
- *include:* to combine schemas from the same target namespace
- *import:* to combine schemas from different target namespaces

In order to coordinate use of the Justice XML schemas with local extension schemas and instances, a concept for assigning, referencing, and using target namespaces should be designed. How to design and arrange target namespaces and reference, include, and/or import the appropriate parts will depend primarily on the data exchange requirements and how exchanges will work (i.e., defined use cases).

There are many ways to structure one or more namespaces for the justice community. This issue relates to using the most appropriate namespace techniques and logical organization.

When defining a standard reusable XML schema, it is usually appropriate to define a target namespace that can be referenced and imported, or included in other XML schemas or instances that need to use its components (without having to redefine them). However, it is not necessary to define a schema as a target namespace. A schema can be anonymous. An anonymous schema takes on the target namespace of its using schema. Declaring an anonymous schema is useful when its contents either have no inherent semantics or define generic structures (for example, a schema that defines a variety of generic list components such as linked lists, vectors, or arrays). The JXDDS is not a candidate for such because it contains components that represent semantically significant objects (persons, vehicles, etc.) and their tag names (elements). Furthermore, the intent of the Justice XML DD is to establish a standard set of justice components, so it makes better sense to define a target namespace for the JXDDS. This will also ensure that the lineage of JXDDS components used by other schemas is maintained and visible.

Whether there should be one or many target namespaces and how they should be organized, relates directly to the reference architecture. Since the justice community is already defining a number of standard XML exchange documents (Rap Sheet, Court Filing, Arrest Warrant, Incident Report, etc.), it is appropriate to define a single *justice target* namespace for the common components that will be used by the other namespaces declared and defined by the exchange documents. This permits the continued creation of data exchange documents without impeding the ongoing development of the JXDDS or Justice XML DD.

Recommendations

Written requirements (or use cases) for the JXDDS do not yet exist. However, through collective experience and discussions on the XSTF, we recommend that the JXDDS schema reference architecture should:

- Implement an object-oriented model of reusable component types and a set of element names (tag names).

- Contain reusable named types to define the structure (content) of standard components so that elements with different semantic meanings but same syntax and structure can be compared and processed in a uniform way.

- Contain standard element names that convey specific semantics in order to:
  - Discourage different element names for the same data concept.
  - Recognize semantically equivalent data.
  - Define standard domain relationships and relate standard elements more easily.

- Have registered target namespaces to promote use of and compliance with standard.

- Have a referenceable schema so that redefinition of components is unnecessary (furthermore, this follows W3C principles and architecture, eliminates redundant schemas, and discourages the creation of unique schema implementations that do not use standard rules for extension).

- Have a well-defined extension mechanism to support its evolution and to enable local specialization for more specific needs.

- Be stored, maintained, and accessible online in a Justice Data Registry and associated Repository. NOTE: A Justice Data Registry/Repository is NOT the same as the much larger Justice Standards Registry/Repository that would store all justice standards. The Justice Data Registry/Repository would be one logical part of and, therefore, inside of the Justice Standards Registry/Repository (or would be a link from it).

- Support the development of standard business context schemas (documents such as Rap Sheet, Court Filing, Arrest Warrant, Incident Report, etc., as well as transactions for query/response).

- Include schema definitions for standard documents (such as ArrestWarrant.xsd, IncidentReport.xsd, RapSheet.xsd, SentencingOrder.xsd, ChargingOrder.xsd, DispositionOrder.xsd, CourtFiling.xsd, etc.)

As explained earlier, there are many ways to organize the schemas and instances to facilitate data exchange. Given the assumptions above, we recommend a reference architecture (refer to Figure 2) that has some advantages in terms of reusability, extension, and maintenance. There certainly may be others. We designate one *justice* target namespace with three logical partitions:

- *standard entity types:* a class hierarchy of object (complex types)
- *standard elements:* all standard tag names (defined as global elements)
- *support types:* required types that do not fit into the hierarchy

The Justice namespace references the namespace of the W3C schema definition. We also designate each standard document (ArrestWarrant.xsd, RapSheet.xsd, etc.) its own target namespace. Each of these documents *imports* the Justice namespace (*import* is used to incorporate components from a different namespace).

Some exchanges will be executed entirely with standard Justice XML DD components and documents, while others may require additional kinds of information objects (extensions to the standard base types) more specific to local activities. We will refer to these as *standard* and *extended* exchanges, respectively.

Standard data exchanges use instances built from the Justice namespace components (by referencing the Justice namespace from the instance), or they can simply employ a standard Justice document specification (that already imports components of the Justice namespace).

Extended data exchanges can also use the Justice standards, but may also import and extend the Justice namespace (or standard document namespaces) to create their own data elements to meet unique local data exchange requirements. Furthermore, if the rules for extending the standard object data model are obeyed (i.e., inheritance is preserved) at the local level, then the locally *extended* schemas will remain compatible with *standard* data exchanges. The reason is that any element whose type has been properly derived from its parent (in accordance with the extension rules) will be valid anywhere one of its ancestor types is expected in an instance. In formal terms this means:

> **If** (element E_1 is of type T_1) **and** (element E_2 is of type T_2)
> **and** (T_2 was derived as a specialization of T_1), **then** T_2 *IS-A* T_1.
> It follows that (E_2 *IS-A* E_1) and, therefore,

If (E_2 *IS-A* E_1), **then** an XML schema parser expecting E_1 will accept and process E_2 as its valid replacement.

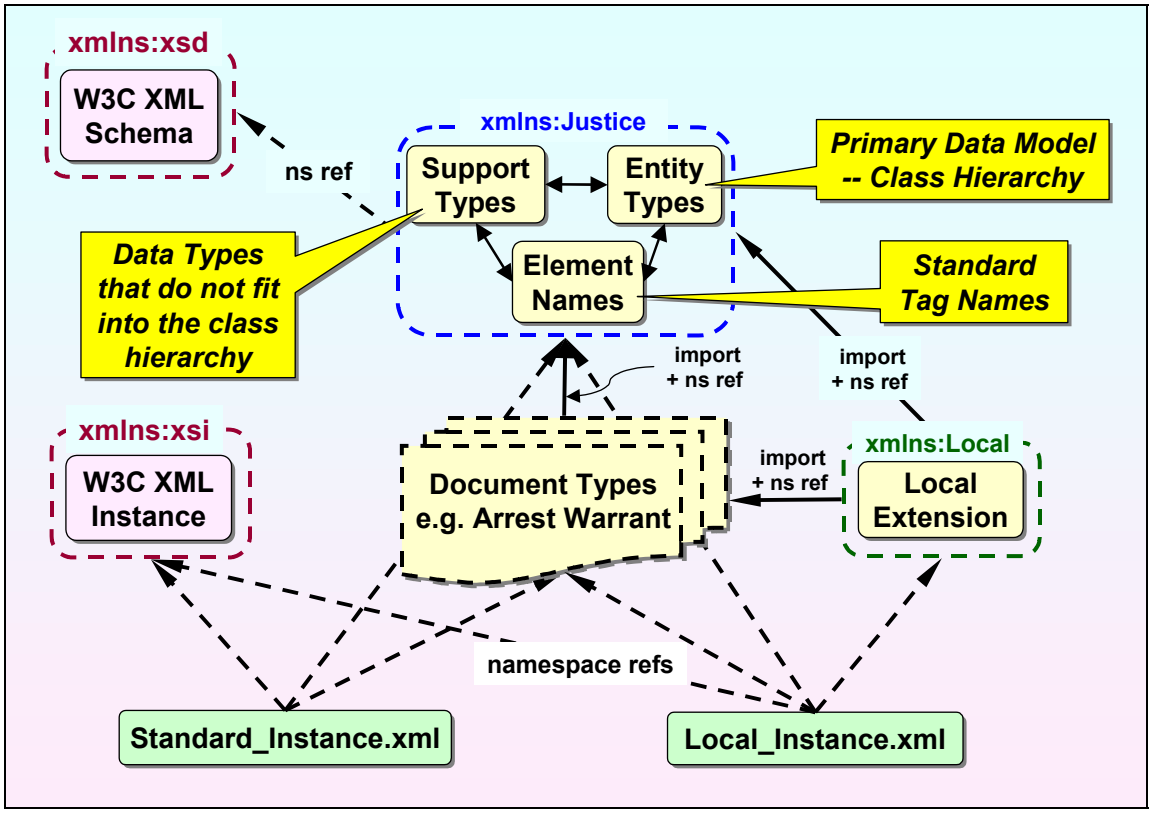

Figure 2. Schema Reference Architecture

In summary, the XSTF supports a single named (not anonymous) Justice XML target namespace. It should include all standard components (both named type and element definitions). Standard exchange documents and other local documents should define their own namespaces and reference or import the JXDDS, as required.

Pros

Provides referential guidance for using Justice XML Standards.

Maintains a logical distinction while allowing operational integration between Justice XML Standards and local extensions.

Cons

There are other possible architectures.

Tends to be document-centric; defines documents outside of class hierarchy.

Examples

Working examples of this referencing architecture can be viewed on the Web at http://justicexml.gtri.gatech.edu (login: justicexml, password: justicegtri). Also, may use http://justicexml:justicegtri@justicexml.gtri.gatech.edu.

Related Questions

How should the JXDDS and associated document schemas be referenced?

- As a local copy (i.e., users maintain a local copy)?
- Online by URI/URL (i.e., one master copy referenced online)?
- As a local cached copy (i.e., local hosts cache and periodically sync with master)?
- Other methods?

---

6. Standard Named Types

What rules/principles determine when to define named types vs. elements?

Discussion / Importance / Impact

In XML schema, types define the structural character of elements (how elements are formed by subcomponents and how those subcomponents are organized). When using a subclassing object-oriented design, types also define the classification of objects and how they are related through the *IS-A* relationship. There are two fundamental ways to define types: anonymously and by name (referred to as *anonymous* types and *named* types). Ordinarily, an anonymous type definition is used to define a single element that will be one of a kind; that is, there will not be any other elements defined with the same type.

Named type definitions are templates for defining multiple elements with the same type. Named types are reusable. Thus, if we are attempting to define a standard Justice XML DD that will encourage reuse of standard components, it is more advantageous to define named types (vice anonymous).

Elements are tag names defined in schemas and used in instances to encapsulate and carry content. Elements can be defined using anonymous type definitions (contained inside the element definition itself) or by using previously defined named types. Although possible to define elements using anonymous type definitions within element definitions, it is rarely useful. Regarding the creation of elements with anonymous types versus named types, the applicable best practice is, when in doubt, define a named type and then reuse that type name to define required elements of that type.

Recommendations

The JXDDS should define and make extensive use of named types.

Pros

Enables the clean separation of type and element definitions.

Named typed definitions can be reused to define elements with same type.

Cons

(none)

Example

The following very simple example illustrates the difference between (1) using a named type definition to define three different elements of the same type, and (2) defining three elements with anonymous type definitions.

(1) Example of one named type definition (*TimePeriod*) used to define three elements (*TimeServed, CaseDuration,* and *PerformancePeriod*) of the same type:

```
...
  <xsd:complexType name="TimePeriod"/>
    <xsd:sequence>
      <xsd:element name="StartDate" type="xsd:date/>
      <xsd:element name="EndDate" type="xsd:date/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="TimeServed"        type="TimePeriod"/>
  <xsd:element name="CaseDuration"      type="TimePeriod"/>
  <xsd:element name="PerformancePeriod" type="TimePeriod"/>
...
```

(2) Example of three elements defined using anonymous type definitions:

```
...
  <xsd:element name="TimeServed">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="StartDate" type="xsd:date/>
        <xsd:element name="EndDate" type="xsd:date/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

```
  <xsd:element name="CaseDuration">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="StartDate" type="xsd:date/>
        <xsd:element name="EndDate" type="xsd:date/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="PerformancePeriod">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="StartDate" type="xsd:date/>
        <xsd:element name="EndDate" type="xsd:date/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
...
```

---

7. Standard Elements

What rules/principles determine which elements should be local and which should be global? Are Justice XML DD definitions precise enough to be unambiguous? For example, *AgencyID* is defined as "Identification information for an agency." *OrganizationID* is defined as "Identification information for an organization." How are these definitions different? Can an agency and an organization be unambiguously distinguished? Should aliasing of standard element names be allowed?

Discussion / Importance / Impact

Earlier, we discussed a schema reference architecture that consists of three logical parts: a class hierarchy of types, standard element names, and supporting types. In this section, we address standard element names.

The issue is not if standard element names are necessary. They are necessary, and the Justice XML DD defines what they should be. The real issue is how they are implemented in the JXDDS. XML schema allows two kinds of element names: *local* and *global*. An element is global and, therefore, visible outside of a schema when its definition occurs in the root of the schema (i.e., as a component of the *schema* element). All other elements (nested inside of other components under the *schema* element) are local and, therefore, not visible or reusable outside their respective containing components.

XML schema's concept of local element names allows the creation of multiple different definitions for the same tag name within a schema as long as each element is contained (nested) within a different component (i.e., locally scoped). The XSTF believes that extensive use of local XML elements is potentially ambiguous and problematic. For example, it will be far more difficult to identify and represent global domain relationships between elements if standard elements are not guaranteed to have a single global definition and meaning. Therefore, all standard element names should be defined globally under the *schema* element. Each use of a standard element within other components of the schema should reference (using the XML *ref* attribute) its global element definition.

The Justice XML DD should have principles that consistently classify elements as global (visible) or local. Examples of such rules might be:

- A global element should prepend the name of the parent object to the element.
- A global element should have globally unique meaning.

Global elements should have names and definitions that have globally recognized semantics. Extensive use of tag names that are generic (such as *date* or *brand*) makes it very difficult to distinguish semantic meaning and identify domain relationships on a global scale. On the other hand, the use of local element names should not be discouraged when appropriate.

Regarding the issue of aliasing, the XSTF believes that allowing aliasing will contribute to ambiguity and hinder the ability to search and compare elements. However, in reality, aliasing of elements will likely happen locally in some unavoidable circumstances. Nonetheless, the policy for the Justice standard should be to strongly discourage aliasing by providing enough flexibility in the standard elements and types to avoid it.

Recommendations

For the data model and associated XML schema implementation, there should be a set of standard global elements defined once at the top level of the schema (as children of the *schema* element), visible to all components of the JXDDS and external schemas. These elements should have names that are as specific as possible (i.e., carry globally semantic meaning in the name). Note that the need for a set of global elements does not preclude the use of local elements when appropriate.

The XSTF also believes that there should be a well-defined set of rules to determine when an element should be global, how to assign an appropriate name, and when it is appropriate to allow local elements.

Global elements should have semantically meaningful names that are as distinguishing and as unambiguous as practically possible. For example, an element called *Date* is not a

good name for a global element. Instead, *CourtAppearanceDate* that is of type *DateType* would be a more appropriately named global element.

The Justice XML DD definition for each element name should be unambiguous and uniquely distinguish each element. ISO/IEC Standard 11179 *Specification and Standardization of Data Elements* provides guidance for the formulation of both standard names and definitions. The Justice XML DD could benefit by consulting this guidance. More review and discussion is necessary in this regard.

No aliasing of element names should be allowed. There should not be multiple ways to encode or structure the same element name. (Note that this should not preclude the need to consider alternative structures for legacy systems or other specific needs; for example, *FullName* and P*ersonName* are semantically the same, but their structures differ to accommodate legacy systems.)

Pros

Defining standard elements at the top level of the namespace schema (i.e., global) has a the following advantages:

- Reduces the potential for ambiguity and ambiguity creep.
- Exposes all names for reuse in future extensions to the class hierarchy.
- Reduces the impact of Justice XML DD extension and change through reuse.
- Ensures that every use of an element refers to one and only one definition and corresponding type; eliminates redefinition of elements.
- Facilitates consistency in searching.

Cons

Defining standard global element names:

- Will require a consistent approach for semantically meaningful names for standard elements, potentially difficult in some cases.
- May require significant changes to current RDD names.
- Prevents any (global) name from being used for another purpose, and may require that many more element names be defined.

Example

The following simple example illustrates several points discussed in this section. One complex type is defined by name: *BuildingType* (detailed definition omitted). This is a global named complex type definition because it is a direct child of the *schema* element. Thus, it can be (re)used any number of times to define elements and types within this schema, or within schemas that use (import or include) this schema, or within instances

that reference the namespace of this schema. Note that several elements in this schema are defined with these types. This schema defines exactly six global elements (children of the *schema* element): *House, Home, Hut, RealProperty, LocalAssets,* and *GlobalAssets*. There are exactly two local elements: *House* (a part of *RealProperty*) and *House* (a part of *LocalAssets*). Note that *GlobalAssets* also has a *House* element; however, its definition is (re)using the global element *House* defined earlier.

Note that this schema defines three distinctly different *House* elements, even though they all have the same identical type definition (*BuildingType*). This is legitimate XML schema and allows three independent *House* entities to exist simultaneously without conflict. If we wanted all three occurrences to represent the same *House* (i.e., one element for *House*), then the *RealProperty* and *LocalAssets* definitions should *ref* the global definition for *House* as *GlobalAssets* does.

This example also suggests the problems associated with permitting aliasing and the need to ensure that Justice XML DD definitions are explicit, unambiguous, and distinguishing. Note that there are at least two cases of potential ambiguity (maybe you can detect more?). There are three global elements that are of *BuildingType: House*, *Home*, and *Hut*. What are the differences? Are they detectable from the names themselves? Should they be? We might assume that a *Hut* is a much smaller version of a *House* or *Home*. However, is there a difference between *Home* and *House*? There may be and, if so, it should be captured unambiguously in the Justice XML DD definitions. Another example is, What are the semantic differences between *RealProperty*, *LocalAssets*, and *GlobalAssets* (note that all have the same structure)? Finally, an example taken from the current RDD definitions is, What is the difference between OrganizationID and AgencyID (to someone without deep domain understanding, such as a developer)?

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>

  <xsd:complexType name="BuildingType"/>
  ...
  <xsd:element name="House" type="BuildingType"/>
  <xsd:element name="Home"  type="BuildingType"/>
  <xsd:element name="Hut"   type="BuildingType"/>

  <xsd:element name="RealProperty">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="House" type="BuildingType"/>
      </xsd:sequence>
    </xsd:complexType>

  <xsd:element name="LocalAssets">
    <xsd:complexType>
      <xsd:sequence>
```

```
        <xsd:element name="House" type="BuildingType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="GlobalAssets">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="House"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Finally, we present one simple example to illustrate the difference between a standard element name that is suitable for local versus one suitable for global status. Recall that we would like to use standard element names that are meaningful in a global sense. Therefore, an element called *Date* is not a good name for a global element (although might be a suitable name for a local element). Instead, *CourtAppearanceDate* that is of type *DateType* would be a more appropriately named global element.

---

8. Support Types

Discussion / Importance / Impact

Referring again to Figure 2 in Section 5 (*schema Reference Architecture and Namespaces*), we defined a single *justice* target namespace with three logical partitions (repeated here):

- *standard entity types:* the class hierarchy of objects (complex types)
- *standard elements:* all standard tag names
- *support types:* required types that do not fit in the hierarchy

We have discussed an XML schema class hierarchy of objects (the logical organization of the primary "things" within the domain) and their complex type definitions with which all standard elements (tag names) will be defined.

There will also be some types that will not fit into the class hierarchy because they are either built into XML schema or perform supporting roles not directly associated with the justice domain. For the JXDDS, these appear to fall into two basic categories: *primitive* types and *package* types.

Built-in XML schema types are in the *primitive* support type category and include string, integer, boolean, date, etc. However, even built-in types can and sometimes must be extended or restricted to conform to specific requirements. The RDD does this with several types including date, time, and string. We believe there are other primitive types that are necessary for the Justice XML DD (for example, amount; i.e., a type for money). Furthermore, we believe there is a finite set of primitive types that classify all required data elements.

In the 1990s, the Department of Defense (DOD) undertook a large effort to standardize data element names and conventions. DOD data administrators developed standards to classify data elements using primitive terms (corresponding to primitive types). These terms are called *class words* (or *property words*).

DOD 8320.1-M *Data Administration Procedures* defines *class word* as the part of a data element name that "designates the category of data into which a data element fits. … establishes the general structure and format of data in the domain for that data element. … categorizes the data at it highest level." Furthermore, "All data elements are required to fit into a category [identified by one class word]. If a new data element does not fit into a category, then a proposal may be made to create a new category [class word]." It is noteworthy that, to date, DOD has not added a single new class word to the original 17 specified. See *Examples* section below for a list of the 17 DOD *class words*.

The 17 DOD class words define a useful set of primitive support types for the Justice XML DD. Several are already included in the RDD; for example, date, time, weight, dimension (height), identifier (a restriction of string), text (essentially an unstructured comment), coordinate (latitude / longitude), and code.

DOD's earlier data standards work appears to be pervasive and has influenced many of today's information standards related to XML for both government and industry. One example is the Universal Data Element Framework (UDEF), a global XML data element indexing scheme that is based directly on the DOD class words. Another example is ISO/IEC Standard 11179 *Specification and Standardization of Data Elements,* which formalizes much of the early DOD work and seems to be a well-referenced standard within much of the OASIS committee work.

The foregoing discussion has focused on the primitive typing role of support types. To illustrate a support type in a packaging role, refer to the *Examples* section below.

Recommendations

The XSTF believes that support types already exist in concept of the Justice XML DD and schema. The discussion above has been an attempt to formalize the concept with guidelines drawn from previous data standards work in DOD that bears a strong relationship to the justice XML effort. The XSTF recommends that more research, review, and discussion take place concerning support types and naming conventions.

Consider the kinds of support types that may be necessary to define simple elements and simple collections of elements. Types that cannot be derived from objects in the class hierarchy may be strong candidates for support types. At a minimum, consider the 17 DOD class words as a starting set of primitive support types. Also, consider the kinds of packaging types (for collections of elements) that may be required to support the architecture.

In this section, we referred to applicable concepts from *DOD 8320.1-M-1 Data Administration Procedures.* This is actually one of a set of manuals from the DOD 8320 series. Since these documents were published by DOD (for its own data admin policies and standards), a more general set of guidelines for developing data dictionaries has been published: *ISO/IEC Standard 11179 Specification and Standardization of Data Elements*. This document captures much of the DOD experience in developing a standard data model, elements, and a dictionary. We recommend that ISO/IEC Standard 11179 should not be ignored by the justice standards effort.

Pros

Mapping every type needed into an object-oriented hierarchy can be very difficult. Using the concept of support types relaxes the requirement to find a place for certain primitive and special types in the class hierarchy.

Requiring one class word in each element name:

- Improves human readability.
- Facilitates semantic mapping to other domains.

Cons

The 17 class words might be too constraining to apply to all justice data elements.

Examples

Examples of useful support types in a primitive typing role correspond to the 17 class words of DOD 8320.1-M-1, selected after many years of experience.

According to Ron Schuldt (a DOD contractor who worked on DOD data standardization and now heads the UDEF effort), these 17 properties enable semantic conflict resolution in a very simple way. Forcing any data element concept to a common starting point (one of 17 properties) appears to be a good way to align data element concepts that are structurally equal but have been assigned different names.

This is a major advantage and the reason for defining named types. Sometimes you will want to compare similar object instances with different semantic meanings (i.e., have different element names) but with the same syntax and structure (i.e., have same type). If they are of the same named type, then comparing or operating on them can easily be done.

It appears that any data element concept property can be mapped into one of these 17 property words. For example, *Velocity* is a type of *Rate* and *PartNumber* is a type of *Identifier*.

The 17 property (or class) words of DOD 8320.1-M-1:

Amount	A monetary value.
Angle	The rotational measurement between two lines and/or planes diverging from a common point and/or line.
Area	The two-dimensional measurement of a surface expressed in unit squares.
Code	A combination of one or more numbers, letters, or special characters substituted for a specific meaning.
Coordinate	One of a set of values that identifies the location of a point.
Date	The notation of a specific period of time.
Dimension	A one-dimensional measured linear distance.
Identifier	A combination of one or more numbers, letters, or special characters that designates a specific object and/or entity, but has no readily definable meaning.
Mass	The measure of inertia of a body.
Name	A designation of an object and/or entity expressed in a word or phrase.
Quantity	A non-monetary numeric value.
Rate	A quantitative expression that represents the numeric relationship between two measurable units.
Temperature	The measure of heat in an object.
Text	An unformatted character string generally in the form of words.
Time	A notation of a specified chronological point within a period.
Volume	A measurement of space occupied by a three-dimensional figure.
Weight	The force with which an object is attracted toward the earth and/or other celestial body by gravitation.

One example of a support type in a packaging type role is a specified container of objects. Recall that in an earlier example (Examples in Section 3 – *Conceptual Data Model Framework*), we derived a *MissingPersonType* from *PersonType*. For example, a type that represents a list of missing persons may be constructed from:

```
<xsd:complexType name="MissingPersonListType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="MissingPerson"
                 type="MissingPersonType"/>
  </xsd:sequence>
</xsd:complexType>
```

Note that the element *MissingPerson* that contains a missing person data type is defined by *MissingPersonListType*, not by *MissingPersonType* itself.

---

9. Data Values, Codes, Constraints, Enumerations

What rules should define how and when to verify acceptable data values with constraint mechanisms such as enumerations, constraint facets, etc.?

How should changes to code tables or acceptable values be handled? (for example, automobile models change every year)?

What standard codes should be adopted (e.g., NCIC, AAMVA, ISO, ANSI, others)? Can, should, or must these be predetermined?

When should the following be used and how should they be implemented?

- Enumerations
- Code table references
- Ranges
- Numeric precision
- XML schema constraints to specify and constrain format of values (for example, the *pattern* facet)

Discussion / Importance / Impact

XML schema provides many mark-up constructs that are useful in constraining the space of values and, therefore, standardizing the format of the data that will be exchanged in XML instances. Some of these include enumerations, cardinality, patterns, ranges, etc. These constructs enable XML validators to check for adherence to correct or standard data formats, thereby relieving applications from this burden. Using such constraints, where appropriate, promotes a higher degree of interoperability by establishing a consistent set of data formats that all exchanges are required to follow. Applications are free to convert or translate content for internal processing or storage but must always send and, in turn, can always expect to receive XML-encapsulated content in the same format.

Enumeration types are a convenient way to constrain values to conform to standard sets of values. However, use of enumeration types for sets that change is bad practice. However, it is important to recognize that even singular modifications or deletions from tables that are implemented as XML schema enumeration types risk invalidating earlier versions of schemas and instances. Therefore, it is safer to use enumerations for extremely stable, discrete concepts that are universally common to a domain; for example, days of the week (Mo, Tu, We, Th, Fr, Sa, Su) or traffic light colors (Red, Yellow, Green). It is tempting to enumerate any value list for which members are never changed or deleted, only occasionally added. However, such a list grows forever and can eventually result in an unacceptably large enumerated type. For example, automobile manufacturers have always created new models periodically. Without deletions, an enumerated list of models would become quite large. This was never the intended purpose for enumeration types that were designed for small, discrete, efficiently processable sets of values.

As the current standard metadata source in widespread use, NCIC 2000 may contain tables that are good candidates for enumeration types. The XSTF believes that the NCIC 2000 tables contain many good concepts for structuring data in XML. The XSTF also believes it is very important to design a standard XML schema method for implementing the metadata in the NCIC 2000 tables.

Recommendations

The XSTF recommends the adoption of *code*, *source*, and *version* attributes for each element whose content will be drawn from standard value lists. The text literal should be the data value itself (for human readability); for example, for element *State* use:

```
<State code="CA" source="NCIC" version="2000">California</State>
```

Extensions should be allowed to standard sources through a mechanism to be specified later. The list of sources should be a small, stable enumerated list. Additions to this enumerated list should be possible through a standard control process for extending the XML schemas. This control process will have to be determined.

The XSTF viewed issues of constraints and representations as very important and worthy of further research and discussion among domain experts and technicians. Several specific value-oriented issues require priority attention and resolution (using standard, consistent methods, if possible):

- Enumeration: What rules should govern when to use enumeration types versus less constraining types?
- Range: How to implement? Example: age between 20 and 25 years old.
- Numeric precision: Example: 5 ft 3 in versus 5 ft
    - How is precision defined?
    - How to represent precision?

- Units: How to represent a distance measured in feet, inches, meters, centimeters, (and conversions for such)?
- Partial values: Assuming this is necessary, how should the capability to exchange partial values be implemented? Example: How to represent the following situation: It is known that a phone number ends in "–3111" but the *AreaCode* and *Prefix* are unknown.
- General null categories: How to represent and unambiguously distinguish the following kinds of values: unknown, none, N/A, nil or null value, etc.?

Pros

Appropriate use of XML schema data value constraint mechanisms:

- Encourages standard, consistent data formats.
- Helps to ensure data integrity XML validation.
- Relieves processing burden from applications.
- Facilitates interoperability.

Cons

Many ways to implement details; need consensus on consistent, flexible standard representations.

Enumeration types can be misused (as discussed above).

Example

Examples are contained in above discussions.

---

10. Representing Metadata

What metadata should the Justice XML DD contain to adequately capture semantic meaning?

Examples of metadata on data element names:

- Definition date
- Base class
- Version number
- Metadata to support/track registry/repository, submission, vetting, legacy data

Examples of metadata on data values (within instances); (i.e., additional XML elements or attributes that encapsulate data values):

- Source of data
- Validity of / credibility of source of data
- Belief in / Trust status of data
- Collection time of data

Who will use the data, see the data, for what purpose, for how long?
Who can copy, file, refile, and redistribute the data?
Records schedules for managing records?
How is 28 CFR compliance handled?
Freedom of Information Act (FOIA) requests?
Similar State FOIA-like regulations (e.g., Georgia Open Records Act)?
Restricted to U.S. jurisdictions?

Discussion / Importance / Impact

There are two kinds of metadata: (1) metadata that describes components in the data dictionary itself, and (2) metadata that describes data values in instance documents. The questions above simply ask if all appropriate metadata has been identified and represented in both categories. The RDD currently defines some metadata items in both categories. For example, elements with time-changing values have a reference date (when last collected).

The discussion at this point tended to wander and ask more questions concerning the kinds of metadata that may be required both within the Justice XML DD and the business exchange documents.

There is a Web site related to publishing information standards called PRISM (www.prismstandard.org) that may be directly related to discussion of these issues. The XSTF should consult this site for applicability

Recommendations

The XSTF believes there is a need to develop a rich set of metadata qualifiers both on the values carried by the components and the Justice XML DD components themselves. This set should be defined and applied to appropriate components in the JXDDS only. Metadata pertinent only to a particular standard business exchange document should be defined in the relevant business document.

In addition, but defined separately from the JXDDS, the effective management of a large Justice XML DD and associated registry/repository requires metadata for tracking the lineage of registered components. Ensure that this metadata is adequately defined for the

Justice XML DD and the registry/repository. Use the OASIS Registry/Repository work for suggestions and guidelines on what metadata may be required.

---

11. Document Container Schemas

Should business container objects be defined for documents and/or transactions (e.g., envelopes in Court Filing)?

What XML objects are necessary to build semantic business containers? Examples include: objects that represent addressee, sender, security, priority, privacy, confidentiality, authentication (digital signature), others (?)

Discussion / Importance / Impact

We distinguish between XML transport container and XML business container objects. SOAP (Simple Object Access Protocol) is a protocol that defines an XML transport container (like an envelope) that can carry an XML business container object (like a business letter or form). A business container usually has associated standard components such as header, signature, security, and privacy tags that overlay business context upon the data/information payload.

An XML business container object can be further classified into two general categories: XML documents and XML transactions. In XML schema terminology, both are considered *XML documents*. However, we will refer to *XML documents* as standard business exchange containers (such as Rap Sheet, Arrest Warrant, Incident Report, etc.) because they are usually *persistent*; that is, they are archived, maintained, or reused intact by either the sender or receiver. Persistent documents usually convey semantic meaning that does not change significantly over time and is strongly dependent on their content being brought together as one entity. Another way of saying this is that documents bring together strongly related objects for some well-defined business purpose or context.

The other database-oriented category of XML documents is *XML transactions.* These may also be predefined for standard XML exchanges. They contain information that is usually assembled by the sender and disassembled by the receiver for the purpose of transmission only. The receiver consumes the data immediately (for example, translates it to his own format and loads his database). There is no expectation that the data in this type of XML exchange will remain together for longer than it takes to transmit and process it. Typically, transactions are transient in nature, although they can be and usually are archived in a transaction log (for backup purposes). They are the XML-tagged data often exchanged by query/response systems.

How does a receiving machine know what to do with an XML document or transaction? In other words, end-to-end applications often expect or require business context (business

related metadata) to understand an exchange and how to process it. There must exist pre-arranged (standard) protocols for transmitting, in addition to the payload data, the appropriate business contextual and procedural information. Analogous to formal paper business correspondence, the payload data may be wrapped in an electronic business *container* or *envelope* built with standard XML components that carry business metadata such as identity of sender and receiver, transmission date, digital signature, security data, priority, etc.

The justice community is already defining standard business containers (Rap Sheet, Arrest Warrant, Incident Report, etc.) and will use standard components from the JXDDS. Therefore, in addition to data payload components, the Justice XML DD and JXDDS should also define common business context containers and components. Figure 3 depicts the example concept of a SOAP transport package that carries an envelope of several standard justice documents. Each document, as well as the envelope itself, consists of several business context components.

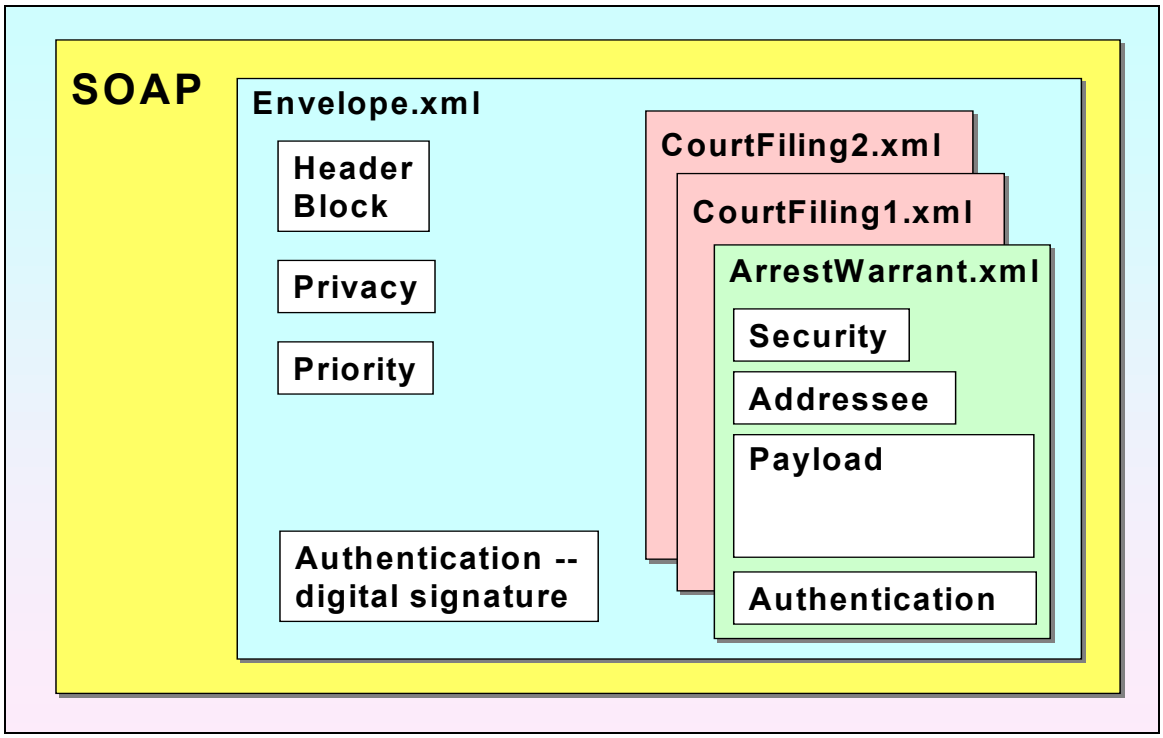

Figure 3. Example Business Containers and Components

Referring back to Figure 2, the reference architecture places the schemas of standard business documents outside of the *Justice* namespace. Each document designates its own target namespace. The reason for this suggested approach is that the justice exchange environment tends to be very document-oriented (exchanges are based on standard archival records and documents such as Rap Sheet, Arrest Warrant, etc.). As such, the standard business documents tend to be large, complex objects that are constructed by

composition. Incorporation of these business objects into the *Justice* class hierarchy as primary objects under a *document* class is another alternative. However, the consequences of this approach should be reviewed and critiqued.

Recommendations

Common components of business container context (including 28 CFR) should be part of the Justice XML DD and JXDDS. This exact set of components should be defined at a later time.

Pros

Entering business context components into the JXDDS will:

- Encourage reuse of business context containers and components.
- Standardize business protocols.
- Facilitate autonomous information exchanges among host machines.
- Enable multiple related document exchanges.

Cons

It might be too difficult to generalize certain business context containers or their components.

Examples

Working examples of a Rap Sheet schema and a corresponding instance built from the JXDDS versions 2.0 and 2.1 are available for review at the following web addresses:

- http://justicexml.gtri.gatech.edu/ (login: justicexml, password: justicegtri)
- http://justicexml:justicegtri@justicexml.gtri.gatech.edu/

Related Questions

Are there or will there be requirements for standard:

- Protocols for XML exchanges?
- Business context containers or envelopes?
- Business container metadata; e.g., headers, priority, filing instructions, index data, security, privacy, digital signature, etc.
- Interfaces to applications or other systems; e.g., case management tools, electronic records management systems, information extraction and link discovery tools, search engines.

12. Component Sizing and Structure

Assuming the JXDDS is built on the reference architecture suggested in Section 5, how are the proper size and structure for the component types in the class hierarchy determined?

Discussion / Importance / Impact

This section addresses component size and structure. However, to discuss the rationales for this section, it is necessary to introduce the concept of relationships, specifically within a class hierarchy. Relationships will be covered in greater detail in Section 13.

We define a *domain relationship* as a relationship between two data objects that is not genuinely represented by an *IS-A* or *HAS-A* relationship. Examples may include *OWNS*, *WAS-SEEN-AT*, *IS-MOTHER-OF*, *IS-ASSOCIATED-WITH*, etc. Sometimes these can be approximated by *HAS-A*, but there is always a potential danger of ambiguity when this is done. Furthermore, this practice builds unnecessarily large, complex objects.

A class hierarchy is built with *IS-A* relationships between type components. The type components themselves are built with *HAS-A* relationships (refer to Figure 2). There are no other explicit relationships in an object model. This means that *HAS-A* is often used to represent implied domain relationships. For example, one may chose to include a person's vehicle in the object for a person, claiming that a person *HAS-A* vehicle. In reality, the relationship is "person *OWNS* vehicle," or perhaps more correctly "person *REGISTERS* vehicle." Neither of these carries the same meaning as "person *HAS-A* vehicle" (persons do not *CONSIST-OF* vehicles). Double-loading the meaning of the *HAS-A* relationship in this way will work if everyone understands the intent. However, if the user community is very large, the use cases are wide in scope, and the number and variety of such implied relationships grows; then there is serious danger, especially if the intent is to establish globally-understood semantics and most of the processing will be handled by machines. Therefore, the XSTF reviewed the use of *HAS-A* in each model in detail.

There are a number of domain relationships represented by *HAS-A* within the Person objects of all models, even the Reconciliation Data Dictionary (RDD). As a result, the PersonType objects of the three models are very large, even though most of the components are optional.

Sizing components appropriately is important if a component repository is to have maximum utility to a large community. Components that are too large (for the sake of capturing "everything anyone might need") are not easily reused. Consequently, many new smaller objects will be created to satisfy more modest needs (for example, *VehicleOwner* in the RDD). It is true that components with optional parts can be reused only for the parts required. However, this practice is dangerous. Suppose that for a given document schema the designer must use a *PersonType* that does not include employment,

residence, or contact information (for privacy reasons). If his only option is a large *PersonType* that includes optional employment, residential, and telephone records, he will be forced to create a new component to avoid risking ambiguity and potential policy violations that result from using a component with options to exchange employment, residence, and contact information.

It is better practice to build small generic *core* components with logically cohesive parts (specified as optional if necessary) and build up to more complex components as needed. This can be done through either subclassing or composition. Subclassing by restriction can also be used to create smaller components from larger components. However, subclassing by restriction is not a good practice because the restricted definition must repeat all the components of the base type definition that are to be included in the derived type (thereby defeating the reuse advantage of subclassing).

It is necessary to define a set of heuristic rules that can be applied to the sizing and structuring of XML schema components. Each rule should also have a rationale for its application (why/when use this rule?). The rationale should (1) identify rule context (when is it or is it not valid to use this rule?) and (2) be based on requirements (or assumed requirements).

In order to understand and prototype the concepts explained above, the XSTF synthesized a prototype *PersonType* using subclassing by comparing three justice data models. This type was chosen because it represents a core object that is central to the justice domain. The data models included:

- The Reconciliation Data Dictionary (RDD) version 1.0.0
- CriMNet: a justice and public safety data model and implementation being designed and built by Mobiam for the state of Minnesota.
- InfoTech: a National Institute of Justice sponsored data model used as the basis for the Los Angeles County California justice information system.

After comparing the *PersonType* component in all three models (RDD, CriMNet, and InfoTech), heuristic rules for sizing and structuring components emerge. The following rules were formed by considering the very large size of the *PersonType* component in all three models, its purpose within those models, and the requirement to reuse it for other purposes. A key observation is that core level components should have generic semantic meaning (i.e., should be able to represent any person, not just a particular kind of person). An extension mechanism should provide the means to specialize it.

Rules for including *HAS-A* parts in *PersonType* will include a part if:

- There is no good rationale to exclude it because it may be useful to some.
- Payload data is contained in an attribute, and then redefine the attribute to an element.

Rules for excluding *HAS-A* parts in *PersonType* will exclude a part if:

- Element clearly represents a domain relationship (i.e., is not a *HAS-A* relationship)
- Element is not an inherent property of the component it is contained in (these are usually domain relationships masquerading as *HAS-A* properties and add to ambiguity). A *HAS-A* property:
    - Should identify the object; or the object should really "consist of" this property.
    - Should be specific enough to this object class (i.e., not too generic and apply to parent).
    - Should not be too specific for this object class (force a subset of the target object).

- Element is a duplication of another element; for example, the InfoTech model uses one element for its own format and another for the corresponding NCIC code. In this case, use one form:

```
<state code="GA" source="NCIC" version="2000">Georgia</state>
```

- Element is metadata, not payload data (convert the element to an attribute).
- Element is actually an extension mechanism; this is probably a candidate for a specific subclass of this object (InfoTech model had this feature).

Some rules are not as absolute, but instead provide guidance for analyzing components. For example, one rational analysis for potential exclusion is:

> If a *HAS-A* property can exist on its own as a useful component (i.e., is reusable), then it should be a component itself. Such a component is probably another complete object with a domain relationship to the component it is attached to (instead of a *HAS-A* property).

Some rationales for logical grouping or structuring of core components or groups in core components:

- Should be a natural result of and implement a true *HAS-A* relationship.
- Should facilitate and encourage reuse.
- Should be cohesive (travel together often).
- Usually have a classification reason for grouping; for example, *PersonType* has properties that are:
    - Stable or permanent (e.g., *DNA*) versus unstable, changing, or changeable (e.g., *HairLength*).
    - Precise or scientific (e.g., *FingerPrint*) versus imprecise (e.g., *BodyBuild*).
    - Visual (e.g., *HairColor*) versus non-visual (e.g., *BloodType*).

Recommendations

XSTF recommends that rules/heuristics for component sizing and structure be designed and applied to future versions of the JXDDS that provide maximum flexibility to users. These rules should guide the construction of top-level core objects that are composed of essential properties common to all objects of that class. Core objects can then be specialized through extension by adding properties (*HAS-A* parts) that distinguish each subclass of the base.

Example

In addition to synthesizing a core *PersonType*, the Task Force took a sample inventory of possible core objects and sub-objects (specializations) that are present in the three models. The following list resulted (not necessarily complete):

- Person
- Organization
    - Agency
- Location (Address, Latitude/Longitude, etc.)
- Contact Information (Phone, Fax, Email, etc.)
- Property
    - Weapon
    - Vehicle
    - Evidence
    - Other
- Incident
    - Accident
- Case
    - Event
    - Conviction

---

13. Relationships Between Components

What commonly recurring relationships exist between components (objects)?

How can these relationships be represented in a standard way?

Will reasoning with the data, metadata, and information be required in the future?

For example, would public safety benefit if law enforcement officials could share both the information objects they maintain as well as the relationships (both known and unknown) that exist among those objects?

How can a catalog of common relationships be formalized and used to search for and automatically identify semantically meaningful links between information objects known to various agencies in different states?

Discussion / Importance / Impact

In the previous section, we introduced the *IS-A* and *HAS-A* relationships and discussed how a class hierarchy is constructed from these. *IS-A* and *HAS-A* impart primitive structural meaning that is often useful in relational databases. We also pointed out that using (or overusing) only the *HAS-A* relationship to represent many different *domain relationships* leads to ambiguity. While humans with deep understanding of the context or domain can often infer correct meaning, it is not possible to create an intelligent program that can autonomously reason with domain relationships that are all represented by *HAS-A*. How is deep understanding of context conveyed? It is conveyed through formal, meaningful representation of domain relationships.

Document containers and business transactions represent a richer, more meaningful set of relationships (e.g., who owns a vehicle; who arrested whom for what reason and when). However, these relationships are localized to these documents and the databases that load them with data. These relationships are generally represented and tracked in databases. What we are most interested in are the relationships that exist between objects in multiple, distributed databases. These are global relationships; they are not usually explicit, rarely visible, and often unknown.

XML schema does not explicitly represent the meaning of global relationships, except as roughly implied by the semantics of content tag names and the positional associations of document structures. These implied relationships and their greater global relevance can often be understood by humans with little effort. However, humans have years of detailed learning, experience, and common sense knowledge of the world, while machines do not. Can such knowledge be represented explicitly and formally for a machine to understand? It cannot be represented with XML schema alone.

Beyond XML schema, the Resource Description Framework (RDF) and higher markup layers of the Semantic Web will facilitate the formal definition of global relationships, common sense domain knowledge, and methods that will enable machines to reason with information exchanged.

Having synthesized and sized core components by applying heuristics that ensure maximum reuse and cohesive structure, it will be necessary to consider the relationships that exist between components. In particular, since we excluded some *HAS-A* parts from the core component (in this case, *PersonType*), we must consider what relationships have been removed and how to either represent them, restore them, or both. Our intention is to globalize as much local meaning as possible. The core components are represented by *IS-A* and *HAS-A* relationships that are unambiguous and globally understood if constructed as discussed earlier. For all other relationships, the following three steps are necessary:

- Identify and define them.
- Classify them.
- Decide how best to represent them.

Clearly, the first step is to know what relationships are required and what they mean. This leads to the need to catalog or classify them. Relationships have properties themselves. Examples include what type of objects they can relate, relative strengths (tight or loose coupling of objects), priority (relate very specialized objects versus core objects), etc.

The following is one set of example relationship classifications discussed:

- Core (*HAS-A*): inherent property; defines the very nature of, consists of, is part of, etc. (NOTE: *IS-A* can also be considered another core relationship, but it is not equivalent to *HAS-A*).
- Fundamental: not core but used very often; almost always transmitted together; tight coupling.
- Domain: not fundamental, but specific; important for particular functions.
- Associations: unspecified or not easy to specify; loose coupling; for example: two telephone numbers are related because they appeared on the same directory listing.

Cataloging relationships is necessary because this can influence how they should be represented. Also, some relationships may not be important enough to promote globally or to represent at all.

Recommendations

The XSTF believes it is fundamentally important to be able to formally represent the relationships. The representation should be open and standard to ensure that commercial tools and technologies can use the relationships. The representation should also be flexible enough to be easily extended, since not all relationships will be identified immediately and new ones will likely be created later. One scheme that provides an open, standard, and flexible representation for documenting relationships in XML is RDF. The XSTF recommends that the W3C development of RDF standards be monitored closely and that RDF be considered for use in the Justice Standards.

Currently, there are several incompatibilities that exist between XML schema and RDF. These include the following:

- All XML schemas define hierarchical (tree-like) data structures. RDF uses XML syntax; however, RDF schemas can define network structures of relationships and metadata.
- RDF does not formally recognize XML complex types.

- In RDF, a URI represents and resolves to a network resource (such as a Web page or a fragment of a Web page) that in turn represents or describes an object. In other words, a URI addresses a network resource that is metadata representing or describing an object. XML schema uses the concept of URI but does not require resolution between a URI and an actual network resource.

The W3C is working to resolve these and other issues. Nonetheless, there are techniques that allow XML schema and RDF to work together. GTRI is exploring such techniques in the context of the justice requirement for representing relationships among XML schema components.

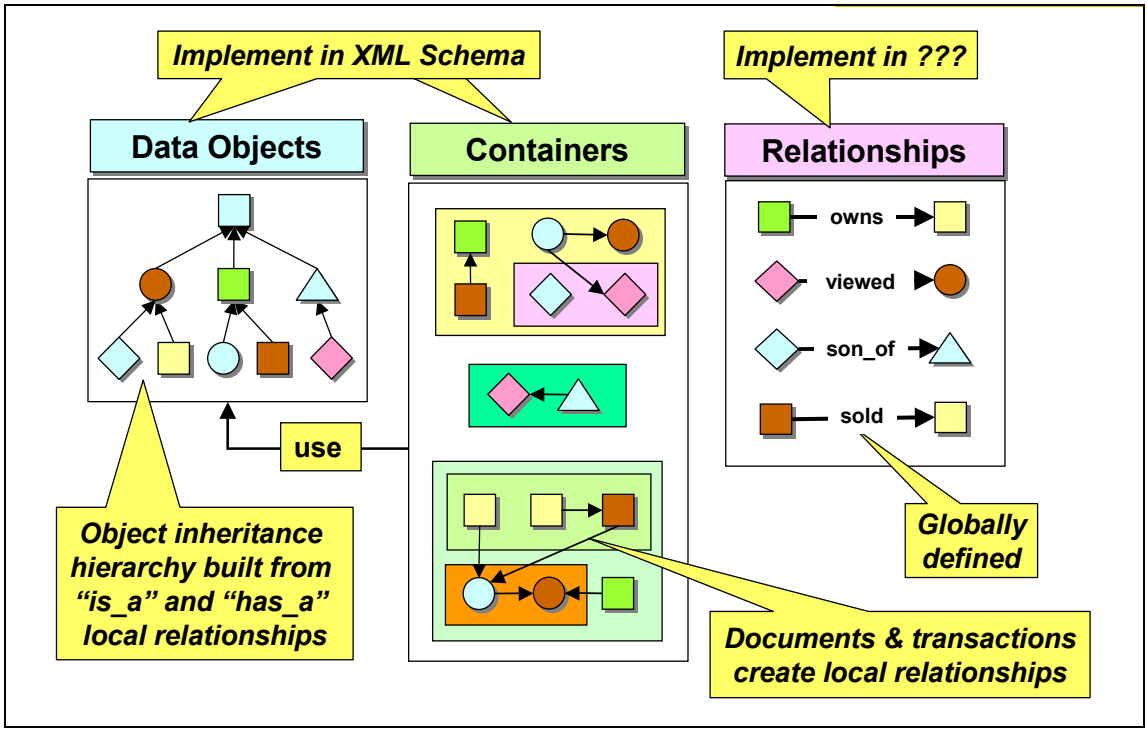

Figure 4. Local versus Global Relationships

Examples

As part of the exercise, the Task Force sampled the space of possible relationships. The following lists resulted. Note that these lists categorize relationships functionally and by objects related (not by the scheme mentioned above). Note also that some of these relationships fall under multiple categories. To capture and understand the semantics of relationships globally requires that these properties be formally represented. This is one purpose of the Resource Description Framework.

Person – Organization	Type of Relationship
affiliated_with	general
works_for	employment
supervised_by	employment
member_of	organizational
leader_of	organizational
customer_of	business
owns	ownership
arrested_by	justice
convicted_by	justice
incarcerated_by	justice
booked_by	justice

Person – Person	Type
affiliated_with	general
seen_with	general
leader_of	organizational
supervised_by	employment
works_for	employment
customer_of	business
business_partner_of	business
arrested_by	justice
convicted_by	justice
incarcerated_by	justice
booked_by	justice
victim_of	criminal
accomplice_of	criminal
father_of	family

---

14. Version Control

To ensure compatibility and application awareness, what is the best way to include a version number within schemas and namespaces?
Assuming Justice XML DD schema evolution, what backward compatibility can be expected or planned? How?

Discussion / Importance / Impact

Applications that use XML and its associated tools to send/receive information should be "aware" of the schema version needed or being used for a particular exchange. This also helps to check compatibility. To do this, it is necessary to employ some form of versioning mechanism for the schemas and instances themselves. There are a number of ways to do this. Some examples have been provided below. Best practices are: (1) to

identify the schema version somewhere inside the schema; (2) to identify within an instance document what schema version(s) the instance is compatible with; (3) to maintain availability of all previous schema versions.

The XSTF discussed version techniques and their importance. However, specific issues of forward and backward compatibility and how to handle them were not discussed. These issues are not necessarily as simple as they may appear. There are trade-offs and dependencies that are not yet well understood. The belief is that a dual approach that uses both internal and external version labels may be necessary (see *Recommendations*). However, it is not at all clear that this is the best solution.

Using an approach that combines both an internal and external version representation, there are potentially six (possibly more) version label locations involved for a single exchange (an internal version number on the schema element and an external version number on its filename for each of three files: the JXDDS, the standard document schema, and its instance). We note that the Draft Federal XML schema Developer's Guide (April 2002) only addresses internal versioning (recommends using the built-in *version* attribute of the *schema* element).

In addition to the examples in this section, two additional approaches to consider for handling versioning were suggested by Mr. Iouri Leonov of Mobiam:

1. For each element in the current Justice XML DD, create a single type and two elements from this type: one element according to the rules of the old Justice XML DD, the other based on a new Justice XML DD. In a *version* attribute, specify which element is which.

2. The use of *substitution groups* might allow one to substitute newer versions for older ones.

Recommendations

The XSTF believes that version information should be provided in two different ways. For external identification, a version should be included in the JXDDS schema file name as well as the schema file names of exchange document schemas. For internal identification, the schema version should be recorded in an attribute within the XML schema itself. However, before it can make a final recommendation, the XSTF also believes that the specific implementation of these techniques should be researched, prototyped, and evaluated to ensure selection of the most practical for the justice environment. This evaluation should also address resulting compatibility issues and associated dependencies.

Pros

Providing version information that an XML validator can verify helps remove this burden from applications.

Cons

Some techniques do not enable XML validators to check and verify version information.

The application of multiple techniques can sometimes cause unexpected results and problems with validation.

Example

Version information can be recorded in:

1. The built-in *version* attribute of the *schema* element.
2. A defined *version* attribute on the root element of the schema.
3. The file name of the schema file (i.e., the target namespace location).
4. The schema's *targetNamespace* element.

The following examples correspond to the four techniques above.

1. Using the built-in (internal) *schema version* attribute:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:justice="http://www.ojp.usdoj.gov/JXDDS"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="1.1.0">
...
</xsd:schema>
```

2. Defining a *version* attribute on the root element:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:justice="http://www.ojp.usdoj.gov/JXDDS"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

<xsd:element name="ArrestWarrant">
  <xsd:complexType>
  ...
  <xsd:attribute name="version" type="xsd:string"
    use="required"/>
  ...
  </xsd:complexType>
</xsd:element>
...
</xsd:schema>
```

The instance corresponding to (2) might begin as:

```
<ArrestWarrant version="2.1.0"
  xmlns:justice="http://www.ojp.usdoj.gov/ArrestWarrant"
  xmlns:xsi="http://www.w3.org/2001/XMLXMLSchema-instance"
  xsi:schemaLocation="http://local.org/ArrestWarrant.xsd">
  ...
</ArrestWarrant>
```

3. Using the filename of the schema (reflected in both the filename of the .xsd file and *schemaLocation* attribute in the root element of the instance; here we illustrate the filename in the *schemaLocation* within an instance):

```
<ArrestWarrant
  xmlns:justice="http://www.ojp.usdoj.gov/ArrestWarrant"
  xmlns:xsi="http://www.w3.org/2001/XMLXMLSchema-instance"
  xsi:schemaLocation="http://local.org/ArrestWarrant.v2.1.0
.xsd">
  ...
</ArrestWarrant>
```

4. Using the schema *targetNamespace*:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:justice="http://www.ojp.usdoj.gov/JXDDSv2.1.0"
  targetNamespace="http://www.ojp.usdoj.gov/JXDDSv2.1.0"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  ...
</xsd:schema>
```

Related Questions

Are there other techniques for version information? Which is best for the justice reference architecture?

---

15. Integration of External Schemas

Should standard data definitions of other organizations (that may have more expertise in a given domain) and/or associated XML schemas be used and integrated? If so, how? By translation or mapping? Adopt one type, the other, or both?

Discussion / Importance / Impact

The current Justice XML DD rule (outlined in the draft Concept of Operations document) defers expertise for particular schema components to the organization that has that expertise. However, XML is a growing technology and currently there is no formal certification process for confirming official domain experts. Therefore, the adoption of external components as standard will likely require test cases. Nonetheless, the W3C anticipated such a diverse, unregulated environment of schemas and naming conventions. That is why they adopted the concept of namespaces.

Recommendations

The decision to use other standards must be addressed on a case-by-case basis. For now, the informal default process for integrating other data definitions into the Justice Standards is the XML namespace mechanism.

Example

If the U.S. Postal Service creates a U.S. Government XML standard *AddressType* that differs from the Justice XML DD *AddressType*, how will this be resolved, used, or integrated with the Justice Standards?

---

16. Related Standards

Discussion / Importance / Impact

There was only brief discussion of these issues. Other standards and standards organizations probably have applicability to the development of a standard Justice XML DD and associated schema. There was not enough information and understanding of the many potentially relative standards for an effective evaluation.

Recommendations

We should consider these standards when appropriate, but not at the expense of derailing our effort. Furthermore, consideration of other standards will require some study and analysis of specific applicability to the Justice XML DD and schema.

Pros

Consulting other standards related to Justice Standards:

- Stimulates new ideas and ways of thinking.
- Often provides some means to compare, measure, and evaluate.

Adopting related standards to guide current work:

- Reuses and gains advantages of previous work by many professionals.
- Improves or sometimes establishes credibility of current work.
- Conforms to and often enables easier integration with similar work in other domains.

Cons

Considering related standards usually requires time to identify, study, and evaluate specific applicability and use within the target domain.

Examples

ISO/IEC Standard 11179 Specification and Standardization of Data Elements [7]:

- Part 1 Framework
- Part 2 Classification of Data Elements
- Part 3 Basic Attributes of Data Elements
- Part 4 Rules and Guidance for Formulation of Data Elements
- Part 5 Naming and Identification Principles for Data Elements
- Part 6 Registration of Data Elements/Basic Attributes of Data Elements

OASIS Registry/Repository and Web Services [10]
Draft Federal XML schema Developer's Guide (April 2002) [6]

---

17. New Technology

What new or future XML-based technologies should be considered and planned for?

Discussion / Importance / Impact

There was only a little discussion on this issue. There was general agreement and understanding that many new Internet and XML-based technologies and associated standards are under development, will be emerging, and are likely to be useful. However, many of these technologies are not yet in widespread use, and so their potential impacts and capabilities are not yet understood in depth.

Recommendations

The XSTF recommends that justice standards groups should monitor, experiment with, and, as appropriate, plan the use of new XML or Internet technologies as they emerge from development and become available in practice.

Pros

Maintains technological currency.

Cons

Usually requires much time to monitor, identify, study, experiment with or prototype, and evaluate applicability of appropriate technologies.

Examples

XML related:

- XSL (XML Style Language).
- Others (http://www.w3.org)?

W3C Semantic Web:

- RDF (Resource Description Framework) for relationships.
- OWL (Web Ontology Language) for meta-relationships and knowledge.

W3C:

- SOAP (Simple Object Access Protocol).
- WSDL (Web Service Description Language).

OASIS Service Web:

- UDDI (Universal Description, Discovery, Integration).

Other

- UDEF (Universal Data Element Framework)

Plan and Recommendations

Although there are still important design and structural issues remaining to review, evaluate, and resolve, the XSTF believes that sufficient understanding of requirements and the domain exists to recommend an action plan. This plan is a logical sequence of steps that quickly establishes an initial baseline information exchange capability from the current Justice XML DD work, thereby preserving investments in time and talent that resulted in RDD version 1.0.0. From this point, subsequent steps apply solutions recommended from the design and structural issues discussed in this report to evolve a more robust standard that enables controlled extension and maximizes reuse and maintenance. We also envision the development of migration tools that will assist users to advance more easily to later versions. As XML technology progresses within the next year, the latter steps will keep pace and eventually harness the full power of RDF to represent domain relationships and potentially enable autonomous reasoning.

Referring to Figure 5 below, the first column represents current XML standards work including the RDD, the XML schema specifications (or Document Type Definitions) used to develop the RDD, their respective instances, and applications that use them. From this base of work moving gradually right in Figure 5, the following recommended actions are detailed in chronological order:

1. Justice XML Data Dictionary Schema version 2.0.0

Action: Develop and release a Justice XML Data Dictionary Schema (JXDDS) version 2.0.0 based on RDD version 1.0.0.

This schema will provide a baseline of reusable components for immediate and rapid development of standard document schemas and transactions (such as Arrest Warrant, Sentencing Order, etc.).

JXDDS v2.0.0 will:

- Consist of normative (standard) schema forms.
- Be referenceable by other schemas and instance documents.
- Correctly validate (using standard XML schema validators).
- Contain named type definitions.
- Contain global element definitions.

A normative, referenceable, valid JXDDS version 2.0.0 with named type definitions and global element definitions based on RDD v1.0.0 is relatively easy to generate. Once deployed, standard schema documents can be quickly and easily created by importing the JXDDS and referencing the *Justice* namespace elements.

This action can be implemented almost immediately, subject to approval and the ability to establish a namespace. Prototypes of the JXDDS v2.0.0 and a streamlined Rap Sheet

schema and instance built from it (based on the *Interstate Criminal History Transmission Specification version 2.2* [1]) can be examined at:

http://justicexml.gtri.gatech.edu/jxdds/XML_Schemas.html
login: justicexml / password: justicegtri

or

http://justicexml:justicegtri@justicexml.gtri.gatech.edu/jxdds/XML_Schemas.html

2. Justice XML Data Dictionary Schema version 2.1.0

Action: Develop and release JXDDS version 2.1.0 based on RDD version 1.0.0.

JXDDS v2.1.0 would follow the *Draft Federal XML Schema Developer's Guide* [6] and use UpperCamelCase (UCC) for types and elements and LowerCamelCase (LCC) for attributes. This would provide consistency and compatibility with current federal policy. JXDDS v2.1.0 would be identical to JXDDS v2.0.0 except for use of UCC.

This action can also be executed almost immediately, subject to the same constraints as the first action. Again, a prototype has already been implemented and is available for review at the Web address given above.

3. Translator Between v2.0.0 and v2.1.0

Action: Develop and release an XSLT (XML Style Language Translation) migration package to translate between XML instances based on JXDDS v2.0.0 and v2.1.0.

This will provide an easy migration path and maintain compatibility between JXDDS versions 2.0.0 and 2.1.0. Because the only differences between JXDDS v2.0.0 and v2.1.0 are lexical (UCC versus LCC), this is a very easy utility to develop and use. A user with a v2.0.0-compatible (LCC) instance document would add a single line to reference the appropriate XSLT file that will translate the instance and present it as v2.1.0 (UCC). The reverse situation works in similar fashion.

We suggest an XSLT package because it is an XML-based solution and can be referenced (and usually executed) directly from within an XML instance. In fact, it is fairly easy to build a translator using a number of other methods, tools, or programming languages. We have implemented both XSLT packages as well as a package using the Practical Extraction Report Language (PERL). These can also be viewed at the Web address given above.

Obviously, this is another action that can be executed almost immediately.

4. JXDDS version 3.0.0

Action: Develop JXDDS version 3.0.0 based on an object-oriented model of reusable core object components and relationships.

JXDDS v3.0.0 will require more time to generate (estimate 6-8 months). This version will require a large amount of design work including decisions and trade-offs concerning the size and structure of components and the cataloging and implementation of relationships. This version would implement a limited RDF capability to enable the representation of domain relationships between data objects.

Projected time of release will depend upon the assumed requirements, issues, principles, and decisions discussed in the *Issues and Findings* sections.

5. Translator for v2.1.0 to v3.0.0

Action: Prototype a translation mechanism to generate JXDDS v3.0.0 from JXDDS v2.1.0 as a migration path to the advanced RDF version.

6. JXDDS version 4.0.0

Action: Develop JXDDS version 4.0.0 that extends version 3.0.0 to a full standards-based RDF (Resource Description Framework) relationship model and representation.

Release is dependent upon the W3C RDF technology development over the next 12 months. We anticipate that JXDDS v4.0.0 could be released within 4-8 months after the release of JXDDS v3.0.0.

7. Translator for v3.0.0 to v4.0.0

Action: Prototype a translation mechanism to generate JXDDS v4.0.0 from JXDDS v3.0.0 as a migration path to the advanced RDF version.

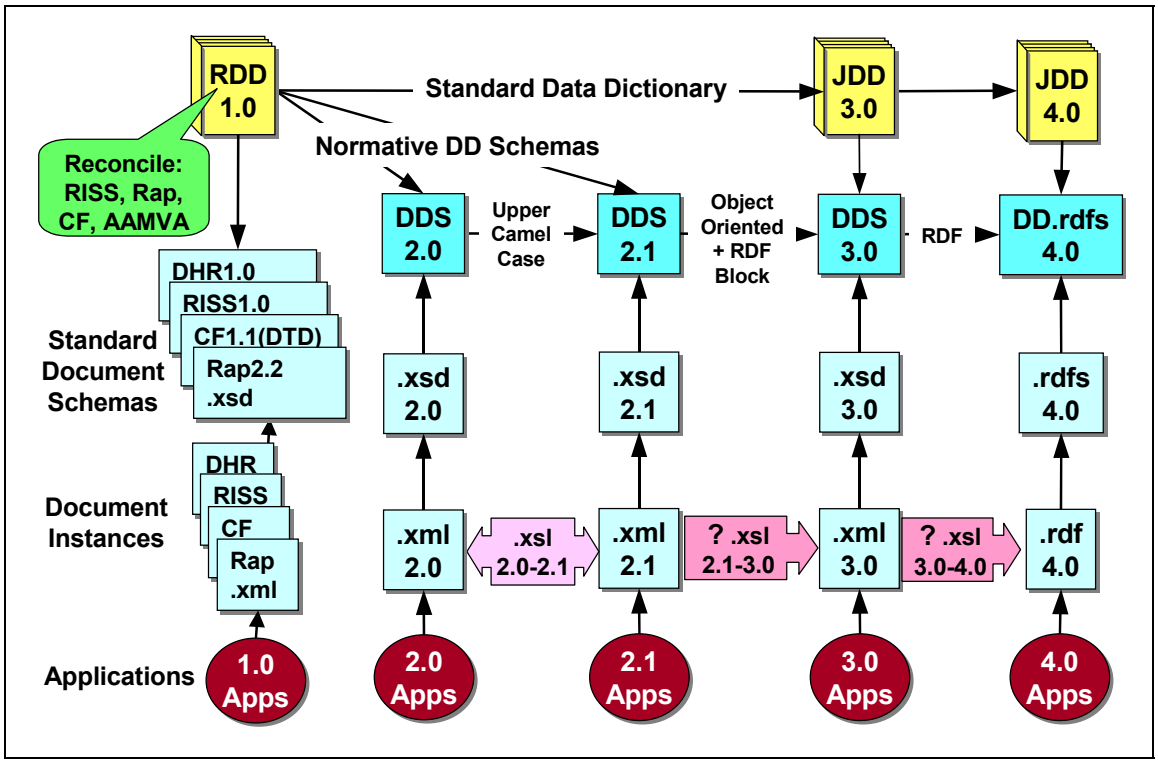

Figure 5. Justice XML Data Dictionary Schema Evolution

References

1. *Interstate Criminal History Transmission Specification (Rap Sheet) XML Version 2.2*, Joint Task Force on Rap Sheet Standardization.

2. *RISS Intel Version 2.0 Database Interface and XML Specification*, Regional Information Sharing System (RISS).

3. *Electronic Court Filing Draft Specification Version 1.1(DTD)*, OASIS LegalXML Electronic Court Filing Technical Committee, 22 July 2002.

4. *Proposed Arrest Warrant XML Document Type Definition (DTD)*, State of Wisconsin (Gerry Coleman), October 2001, http://www.doj.state.wi.us/les/XML/files/

5. *XML Justice Data Dictionary Standard (Draft)*, Infrastructure and Standards Working Group (ISWG) of Global Justice Information Network (GJIN), October 2001.

6. *Draft Federal XML Schema Developer's Guide*XML.gov, April 2002, http://xml.gov/documents/in_progress/developersguide.pdf.

7. *ISO/IEC Standard 11179 Specification and Standardization of Data Elements,* International Standards Organization (ISO) / International Electrotechnical Commission (IEC).

8. *Principles of XML Development for Justice and Public Safety*, Infrastructure and Standards Working Group (ISWG) of Global Justice Information Network (GJIN), National Telecommunications and Information Administration, (working document).

9. Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/guidelines.shtml

10. *OASIS Registry/Repository Technical Specifications* (Working Draft Version 1.1), Organization for the Advancement of Structured Information Standards (OASIS), December 2000, ftp://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf

Acronyms

28 CFR	Title 28 - Code of Federal Regulations
AAMVA	Association of American Motor Vehicle Administrators
ANSI	American National Standards Institute
DD	Data Dictionary
DHR	Driver History Record
DOD	Department of Defense
GTRI	Georgia Tech Research Institute
IEC	International Electrotechnical Commission
ISO	International Standards Organization
ISWG	Infrastructure and Standards Working Group
JXDDS	Justice XML Data Dictionary Schema
LCC	Lower Camel Case
MS	Microsoft
NCIC	National Criminal Information Codes
NIST	National Institute for Standards and Technology
NTIA	National Telecommunications and Information Administration
OASIS	Organization for the Advancement of Structured Information Standards
OWL	Web Ontology Language
PDF	Portable Document Format (Adobe® File Format)
RDD	Reconciliation Data Dictionary
RDF	Resource Description Framework
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UCC	Upper Camel Case
UDDI	Universal Description, Discovery, & Integration
UDEF	Universal Data Element Framework
URI/URL	Universal Resource Identifier / Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSL	XML Style Language
XSTF	XML Structure Task Force