

# Knowledge Representation for Distributed System

Jinghai Rao

January 15, 2001

## Abstract

There is an increasing demand for formalized knowledge on the Web. Several communities are getting ready to offer semiformal or formal Web content. XML-based markup languages provide a 'universal' storage and interchange format for such Web-distributed knowledge representation. I will try to show how to map AI representations (e.g., logics and frames) to XML (incl. RDF and RDF Schema), discuss how to specify XML DTDs and RDF (Schema) descriptions for various representations, survey existing XML extensions for knowledge bases/ontologies, deal with the acquisition and processing of such representations.

## 1 Introduction

The current Web begins to use the Extensible Markup Language (XML)[8] to encode information and services with meaningful structure and semantics that computers can readily understand. Several communities are getting ready to offer semiformal or formal Web content. In Agent system, the agents must adopt common ontologies if they are to interact without misunderstanding. Many agent systems also use XML as the knowledge representation language. So the research on how to use XML to represent knowledge and share common ontology is very important.

In this paper, I present how XML-based markup languages provide a 'universal' storage and interchange format for such Web-distributed knowledge representation. In section 2, I briefly introduce the concept and technical description of XML and the currently available standards for semantic interoperability, such as XML/EDI and RDF. In section 3, I give the concept of the word "ontology" and how to share the ontology with XML. The section 4 is about using markup language in knowledge representation. In this section, I compare the two leading techniques, XML and RDF. Finally, the last section summarizes and concludes the article.

## 2 Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a W3C Recommendation for marking up data that cannot be marked up using the HyperText Markup Language

(HTML). XML is a simple dialect of the Standard Generalized Markup Language (SGML) defined in ISO Standard 8879. The goal of XML is to enable SGML-coded data to be served, received, and processed on the Web in the way that is as easy as that currently made possible by use of the fixed SGML tag set provided by HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. XML is based on the ISO 10646 Universal Multiple-Octet Coded Character Set (UCS) so that it can be used in all major trading nations.[5]

The existing XML standard will be complemented by two extensions. Part 2 of the standard will permit XML documents to be linked together. Part 3 will provide a method for specifying the presentation style, and controlling the behaviour, of XML elements.

One area where XML is anticipated to be particularly important is in the area of electronic commerce. Traditional mechanisms for electronic data interchange(EDI) are based on the interchange of compactly codes messages between the computer systems of two or more businesses. But traditional EDI is complex and expensive. Moreover, EDI's brittle syntax necessitates a custom integration solution between each pair of trading partners. For these reasons, EDI transactions will increasingly take place over the Internet using an XML/EDI message format. This is really a promising trend for XML technology.

Early in the development cycle XML was identified as a natural encoding format by those attempting to work out how to exchange metadata about stored objects. After a number of supplier-specific proposed solutions had been considered, on solution, the Resource Description Framework (RDF), became the clear favourite. RDF is used to provide an XML-coded definition of metadata languages which can be used to exchange sets of metadata.

## 2.1 Technical Description

The Extensible Markup Language (XML) is a W3C Recommendation for marking up screen presentable data that cannot be marked up using the simple set of document presentation elements defined in the HyperText Markup Language (HTML). Whilst designed initially for the display of documentation distributed via the World Wide Web (WWW), XML has been widely adopted as a means of interchanging information between computer programs. In particular it is widely seen as the best solution for the interchange of metadata about stored objects and programs (e.g. the Open Software Description) and for the interchange of commercial information (e.g. Open Financial Exchange).

XML is an extremely simple dialect of the Standard Generalized Markup Language (SGML) defined in ISO Standard 8879. SGML was designed in the 1980's as a tool to enable technical documentation and other forms of publishable data to be interchanged between authors, publishers and those responsible for the production of printed copies of data sets. By providing a formal definition of the component parts of a publishable information set, SGML made it possible to verify the correct transmission and receipt of interchanged data sets. It was soon found that these techniques are applicable in areas other than those

directly related to publications. For example, SGML is often used as a neutral data format when moving data between databases as part of multinational projects.

The goal of XML is to enable SGML-coded data to be served, received, and processed on the Web in the way that is as easy as that currently made possible by use of the fixed SGML tag set provided by HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. Unlike early versions of SGML and HTML, XML has been based from the very start on the ISO 10646 Universal Multi-Octet Coded Character Set (UCS, which includes the codes that make up the Unicode character set) so that it can be used in all major trading nations.

An XML document instance must be created and stored as a set of properly nested data storage entities, each of which is made up of a number of logical elements which contain data or define processes to be performed. The outermost storage entity is referred to as the document entity: it contains both the start and the end of the root or document element of the document instance. Elements can be nested to create hierarchies (information trees), and may contain references to embedded entities. Elements can be assigned attributes (properties) which indicate how the contents of the element should be interpreted.

Each XML element starts with a named start-tag and ends with an end-tag with a matching name. Outward pointing angle brackets are used to delimit these markup tags (e.g. < title > ). An end-tag is distinguished from a start-tag by having a slash immediately preceding the name (e.g. </title>). Elements that have no contents are distinguished by having a slash immediately after the name in the start-tag to indicate that the end-tag has been omitted (e.g. <image/>). Because each element of an XML document has clearly marked limits, it is easy to determine when its contents have been received over a network.

Attributes of XML elements are defined as part of its start-tag (e.g. <image title="Front view" source="entity21"/>). Each XML attribute must be fully defined, with the attribute name followed by a value indicator (=) and a quote delimited string containing the attribute value. Attributes can be assigned a default value if an attribute list declaration is associated with the formal declaration for the element in the document type declaration (see below).

XML requires that data that is not coded in XML characters be stored in a named binary entity, which may have associated with it the name of the notation in which its contents have been encoded. The location and notation of each uniquely named binary entity must be declared in an entity declaration (e.g. <!ENTITY entity21 SYSTEM "http://www.myco.com/figs/figure1.gif" NDATA GIF>). The location of a processor for a notation can (optionally) be identified using a notation declaration (e.g. <!NOTATION GIF SYSTEM "show-gif.dll">). XML uses Internet Uniform Resource Locators (URLs) to identify locations of external entities and other types of files. Relative URLs can be used to identify locally stored information.

Parts of an XML document instance can be stored in separate files that will be referenced as external text entities. Such entities are declared in the

same way as binary entities, except that there is no associated notation name. Alternatively internal text entities can be used to define the replacement text for an entity reference. For example, addition of an entity declaration of the form `<!ENTITY company "The SGML Centre">` to a document type declaration will allow an entity reference of the form `&company;` to be entered in the associated document instance. This reference will be replaced by the quoted replacement text defined in the entity declaration when the file is processed (parsed).

The set of elements, attributes, entities and notations that can be used within an XML document instance can (optionally) be formally defined in a document type definition (DTD) that is associated with the document instance through the addition of a document type declaration that forms part of the prolog of a document instance. The declarations that make up the document type definition can form part of a file referenced as the external subset of the document type declaration, or can be embedded, or referenced, within the internal subset of the declaration. Comment declarations must be used for any explanations required as part of the document type definition.

XML document type definitions inherit a set of predefined entity definitions that provide names for the characters used to delimit XML markup. Each of these names maps to a numeric character reference that defines the decimal value of the ISO 10646 code point for the associated delimiter character. For example, the predefined entity whose reference has the form `&lt;` maps to decimal character reference `&#60;`, which identifies the `<` symbol in the ISO 10646 code set.

Where there is no formal declaration for an element, attribute or notation in the document type definition, an XML document processor (XML parser) will apply a set of default rules for determining how to process the markup. To indicate that documents are to be processed according to these default rules, using predefined entity sets, an XML document instance, and any associated document type declaration, must be preceded by a processing instruction indicating which version of XML the document conforms to, and what character encoding was used to create the storage entity. By default this XML declaration will take the form `<?xml version="1.0" encoding="UTF-8"?>`. This information can be supplemented by a statement indicated whether or not the document is self-contained (contains its own document type definition) by the addition of `standalone="yes"`.

Because XML is a subset of SGML, parsed document instances can be referenced using groves of the form defined in the SGML Extended Facilities annex of ISO/IEC 10744. Alternatively the Document Object Model (DOM) defined by W3C for use by both HTML and XML can be used to identify the structure of an XML element or entity tree. Applications requiring a simpler application programming interface (API) can use the event-based Simple API for XML (SAX) developed by the XML Developers Group. Both DOM and SAX have IDL definitions that allow XML elements to be stored in CORBA compliant databases.

## 2.2 Typical Applications

### 2.2.1 XML/EDI

One area where XML is anticipated to be particularly important is in the area of electronic commerce.[3] Traditional mechanisms for electronic data interchange (EDI) are based on the interchange of compactly codes messages between the computer systems of two or more businesses. Each message has to be decoded before its contents can be processed or presented to users. Web-based commerce has, by contrast, been based on the concept of completing an HTML form and then posting the results back to the initiating server for processing, without any details of the transaction being retained by the company completing the form. Neither of these approaches allows for a fully integrated approach, where exchanged data forms part of a business cycle that starts with product marketing, proceeds on to contract negotiation, purchasing, delivery logistics management, payment and tax and other administration data. It is anticipated that a combination of the behaviour control mechanisms of XML links and the client-side evaluation functionality that can be provided by XSL may make it possible to develop message exchange systems where data can be captured on screen, processed locally as required, transmitted to the relevant receiver(s) for processing as relevant without having to reformat the data during the processing. An early example of such a protocol, called the Internet Open Trading Protocol (OTP), was published in draft form in January 1998. This protocol is designed to handle messages between consumers and merchants trading over the Internet. It also allows for the use to third parties such as value acquirers, deliverers and customer care providers.

### 2.2.2 RDF

RDF[7] is a recent W3C recommendation, and as a result less widely known than XML. Its motivation is to provide a standard for meta-data, for descriptions about resources on the web. However, the distinction between data and meta-data is sophisticated: RDF as framework for describing data about web-resources is also capable of representing data.

The basic construction in RDF is an object-attribute-value triple: an object O has an attribute A with value V. Such a triple corresponds to the relation that is commonly written as A(O,V), for example: hasPrice('http://www.books.org/ISBN0012515866', "\$62"). A third way to think about such a basic RDF triple is as a labeled edge between two nodes: '[O] -A -> [V]'. This last notation is particularly useful, since RDF allows objects and values (1st and 3rd elements of the basic RDF triples) to be mixed: any object can play the role of a value.

Furthermore, RDF allows a form of reification, allowing any RDF statement to be the object or value of a triple allowing the graphs not only to be chained, but also to be nested. Finally, it is possible to indicate that a given object is of a certain type, such as stating that 'ISBN03547X' is of type book:

```
< rdf:Description about="www.books.org/ISBN0012515866">
```

```
< rdf:type resource=http://description.org/schema/book>
< /rdf:Description>
```

It is important to note that the intended role of RDF is to provide a basic object-attribute-value (OAV) data-model for meta-data. Besides this intended Object-Attribute-Value-semantics (which is itself only informally described in the RDF standard), no further data modeling commitments are made. In particular, no particular reserved terms for any further data modeling are defined. Like the XML data model, the RDF data model provides no mechanisms for declaring the property names that are to be used.

Just as XML Schema provides a vocabulary definition facility for XML, RDF Schema provides a similar facility for RDF. RDF Schema allows the definition of a particular vocabulary that should be used for RDF attributes (e.g. `authorOf`), and it allows the specification of the kinds of object to which these attributes may be applied. In other words, the RDF Schema mechanism provides a basic type system for use in RDF models. This type system itself uses some predefined terminology. These are terms such as `Class` and `subClassOf` that are used in specifying application-specific schemas. RDF Schema expressions are themselves also valid RDF expressions (just as XML Schema expressions are valid XML).

Two crucial RDF Schema constructions are `subClassOf` and `subPropertyOf`. RDF objects may be instances of one or more classes; this is indicated using the type property. The `subClassOf` property form RDF Schema allows the specification of the hierarchical organization of such classes. `subPropertyOf` does the same for properties. Furthermore, constraints on properties can be specified using domain and range constructs.

### 3 Share the Ontology with XML

The word "ontology" [6] seems to generate a lot of controversy in discussions about AI. It has a long history in philosophy, in which it refers to the subject of existence. It is also often confused with epistemology, which is about knowledge and knowing.

In the context of knowledge sharing, the term ontology is used to mean a specification of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy.

An ontology is a specification used for making ontological commitments. The formal definition of ontological commitment is given below. For pragmatic reasons, we choose to write an ontology as a set of definitions of formal vocabulary. Although this isn't the only way to specify a conceptualization, it has some nice properties for knowledge sharing among AI software (e.g., semantics independent of reader and context). Practically, an ontological commitment is

an agreement to use a vocabulary (i.e., ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology. We build agents that commit to ontologies. We design ontologies so we can share knowledge with and among these agents.

A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.

An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.<sup>1</sup>

We use common ontologies to describe ontological commitments for a set of agents so that they can communicate about a domain of discourse without necessarily operating on a globally shared theory. We say that an agent commits to an ontology if its observable actions are consistent with the definitions in the ontology. The idea of ontological commitments is based on the Knowledge-Level perspective. The Knowledge Level is a level of description of the knowledge of an agent that is independent of the symbol-level representation used internally by the agent. Knowledge is attributed to agents by observing their actions; an agent "knows" something if it acts as if it had the information and is acting rationally to achieve its goals. The "actions" of agents—including knowledge base servers and knowledge-based systems— can be seen through a tell and ask functional interface, where a client interacts with an agent by making logical assertions (tell), and posing queries (ask).

XML makes it easy to create specialized markup languages that identify and describe the interaction on the Internet. For example, in the Electronic Commerce, if every business invented its own XML definitions for product catalogs,

---

<sup>1</sup>Ontologies are often equated with taxonomic hierarchies of classes, but class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions, that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world. To specify a conceptualization one needs to state axioms that do constrain the possible interpretations for the defined terms.

requests for quotes, price lists, purchase orders, invoices, and so on, the Web would be scarcely more usable as a platform for agents and other automated processes than it is today.[2] Agent-based systems must adopt common ontologies if they are to interact without misunderstanding. Realizing Web automation in such complex environments reopens many of the problems and issues the knowledge-sharing and intelligent agent communities have been wrestling within such initiatives as the shared design environment. Some platforms have been developed, such as SHADE or ATOS, to enable agents working on different problems to share ontologies and interoperate over networks.

## 4 Representing Knowledge with Markup Language

The Web is a universal medium for exchanging data and knowledge: for the first time in history we have a widely exploited many-to-many medium for data interchange. XML offers new general possibilities, from which AI knowledge representation (KR) can profit:[1]

1. Definition of self-describing data in worldwide standardized, non-proprietary format
2. Structured data and knowledge exchange for enterprises in various industries
3. Integration of information from different sources (into uniform documents)

This medium poses new requirements for any format used for exchanging data on the web:[4]

1. Universal expressive power: Since it is not possible to anticipate all potential uses, a data format must have enough expressive power to express any form of data.
2. Support for Syntactic Interoperability: By syntactic interoperability we mean how easy it is to read the data and get a representation that can be exploited by applications. For example, software components like parsers or query APIs should be as reusable as possible among different applications. Syntactic interoperability is high when the parsers and APIs needed to manipulate the data are readily available.
3. Support for Semantic Interoperability: With semantic interoperability we mean the difficulty of understanding the data. Please note the difference from syntactic interoperability: syntactic interoperability talks about parsing the data, while semantic interoperability means to define mappings between unknown terms and known terms in the data. We regard this requirement as one of the most important, since the cost of establishing semantic interoperability is usually higher than for establishing syntactic interoperability, due to the need for content analysis.

In the next two subsections we compare XML and RDF regarding the above sketched requirements.

## 4.1 Using XML

XML is just formalism for defining a grammar, so anything can be encoded in XML if a grammar can be defined for it. This means that the first requirement is fulfilled by XML.

Since an XML parser can parse any XML data, and is usually a reusable component (XML parsers are already part of standard software libraries.), the second requirement is also fulfilled by XML.

However, when it comes to semantic interoperability, XML has disadvantages. XML is aiming at the structure of documents and does not impose any common interpretation of the data contained in the document. This is the major limitation of XML: since XML just describes grammars there is no way of recognizing a semantic unit from a particular domain of interest.

Although this limitation is at the heart of the “schema-wars” which are currently raging at forums such as [www.biztalk.org](http://www.biztalk.org), [www.oasis-open.org](http://www.oasis-open.org), [www.rosettanet.org](http://www.rosettanet.org), [www.schema.net](http://www.schema.net), etc., this fundamental aspect of XML is not yet widely recognized.

We illustrate this point by two scenarios. In the first scenario (figure 1), two applications try to communicate to each other. Both applications agree on the use and intended meaning of the document structure given by DTD A.

However, before data exchange can be established between the applications, a model of the domain of interest has to be built, since it is necessary to clarify what kind data is sent from the first application to the second. This model is usually described in terms of objects and relations (e.g. like in UML or Entity Relationship Modeling). From the domain model a DTD or an XML Schema is constructed. Since a DTD just describes a grammar, and there exists multiple possibilities to encode a given domain model into a DTD. So the direct connection from the DTD to the domain model is lost and it cannot be easily reconstructed.

The second scenario (sketched in figure 2) occurs frequently on the Web: new business partners have to be added to an existing relationship, new information sources become available etc. Since this is a very frequent operation, it is important to reduce the costs of adding new communication partners as much as possible. However, using XML and DTDs (or XML Schemas) requires much more effort than necessary: it is not sufficient to map one domain model to the other one, since the domain models are encoded in the DTDs, and have to be reengineered first. A direct mapping based on the different DTDs is not possible, since the task is not to map one grammar to another grammar, but to map objects and relations from one domain of interest to another one. So we have to define the mappings between the different domain models, not between the different DTDs. Defining the mapping between DTDs is an additional step, after the correct domain mappings are identified. So the following tasks have to be performed:

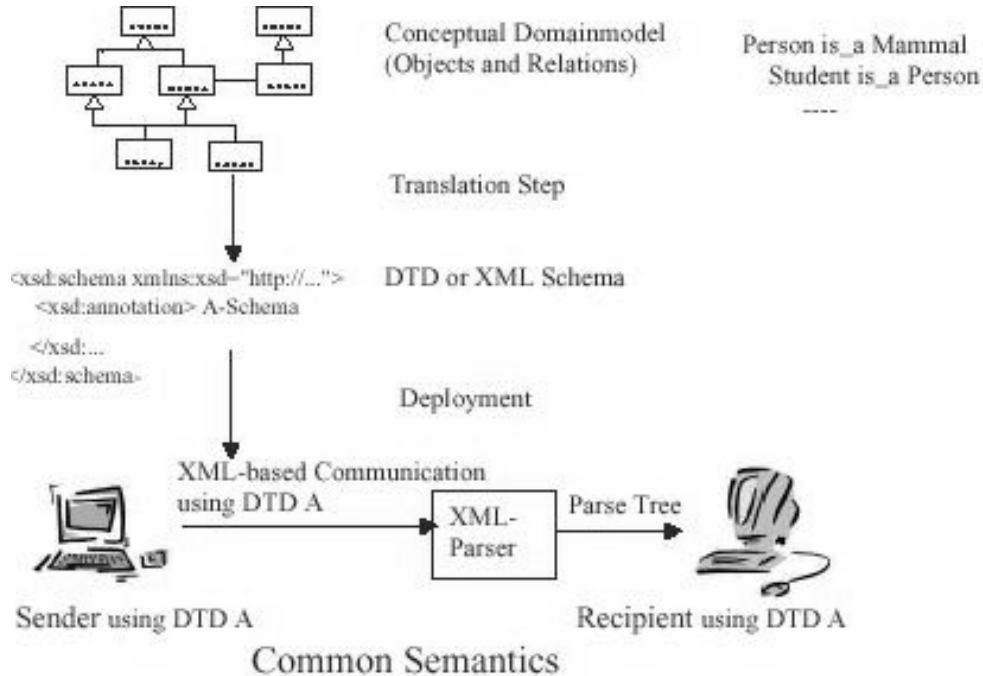


Figure 1: Two application share the same DTD

1. Reengineering of the original Domain Model from the DTD or XML Schema. Since it is necessary to map semantic entities (and not grammars) it is necessary to reconstruct the original domain models that was used to construct the DTDs.
2. Establishing mappings between the entities in the domain model: the concepts and relationships from the domain models have to be mapped to each other. Here techniques developed in the Knowledge Engineering and Database area are often helpful.
3. Defining translation procedures for XML Documents: since XML documents are exchanged, the mappings established in the task 2 above have to be translated in mapping procedures for XML documents (e.g. XSLT definitions for grammars). This is again a high-effort task, since it depends on the particular encoding chosen to construct the initial DTDs.

Although step (2) is already non-trivial, additional effort has to be spent in the particular encoding. The additional effort consists of: the translation of the original domain model into an XML-DTD, the reengineering of the domain model, and the generation of mapping procedures for XML documents based on

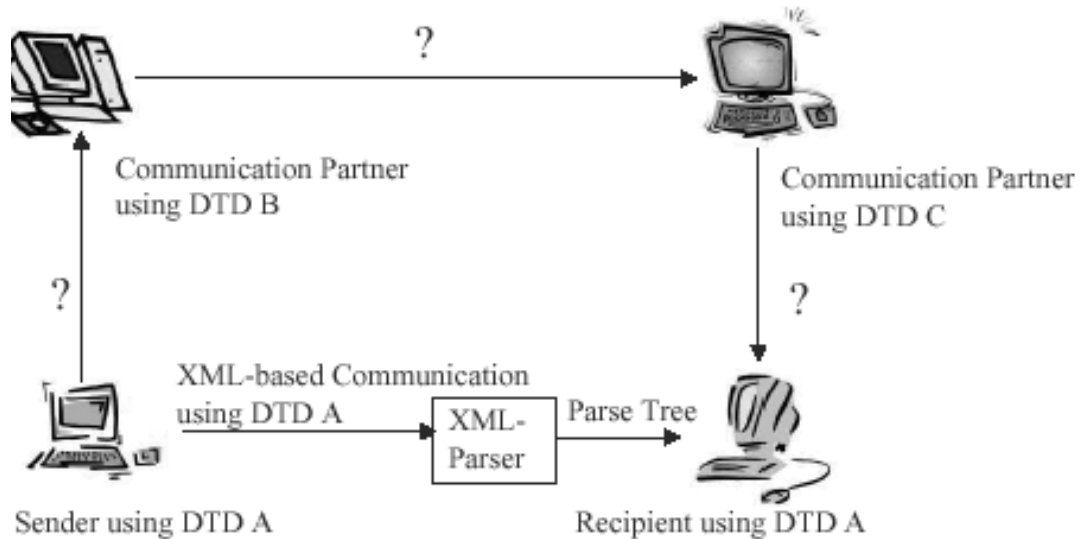


Figure 2: Two application share the same DTD

the established domain mappings. Using a more suitable formalism for the data transfer than pure XML can save much of the additional effort, since then the translations are not necessary anymore. In a nutshell, this means that XML is very suitable for data interchange between applications that both know about what the data is, but not in situations where the addition of new communication partners occurs frequently.

## 4.2 Using RDF

RDF defines a nested object-attribute-value structure. When it comes to Semantic Interoperability, RDF has significant advantages over XML: semantic units are given naturally through the object-attribute structure: all objects are independent entities. A domain model, defining objects and relationships of a domain of interest, can be represented naturally in RDF, so translation steps, as required when using XML, are not necessary. To find mappings between two RDF descriptions, techniques from Knowledge Representation are directly applicable. Of course this does not solve the general interoperability problem, that is, finding semantic-preserving mappings between objects. However, the usage of RDF for data interchange raises the level of potential reuse much beyond parser reuse, which is all that one would obtain from using plain XML.

Furthermore, since RDF describes a layer independent of XML, the RDF model (and software using the RDF model) can still be used, even if the current XML syntax is changed or disappears.

Another way of phrasing the advantages of RDF is as follows. Ideally, we

would like a universally shared Knowledge Representation language to support the Semantic Web. For a variety of pragmatic and technological reasons, this ideal is not achievable in practice, and we will have to live with a multitude of meta-data representations. RDF contains as much Knowledge Representation technology as can be shared between widely varying meta-data languages. Furthermore, the RDF Schema language is powerful enough to define richer languages on top of the limited primitives of RDF.

For example, if we are concerned with exploiting the semantics of Web-page contents, we should consider the two main approaches in Computer Science to modeling semantics: declarative and procedural semantics.

In a declarative semantics, the meaning of an expression E is given by a mapping to another, well-understood formalism, or by stating the conclusions or properties that follow from E. The meaning of expression E can be understood without reference to any specific computational procedure, which is why this approach to semantics is dubbed "declarative". Using procedural semantics, the meaning of an expression E is given by referring to the behavior that some real or virtual procedure (or program, or machine) will exhibit on E. Often the only way to obtain the meaning of an expression E under such a procedural semantics is to simply execute the procedure on E, and observe its behavior.

This difference between a declarative and a procedural semantics loosely coincides with the difference between the XML and RDF approaches to semantics of Web-pages. As we've argued, an XML expression has no inherent semantics, and its semantics is only determined by the actions that one or more programs undertake on the XML expression (e.g. is tag-nesting interpreted as part-of, or subtype-of, or something else again?). An RDF expression on the other hand has a specific declarative semantics (e.g. the intended meaning of "subClassOf"), and this is specified independent of any processor for RDF expressions (or stated otherwise: any RDF processor must conform to this intended semantics).

Together with the W3C, we stand in the line of along tradition in Computer Science and AI, which argues that the declarative approach to semantics leads to more shareable and extendible information and knowledge sources. We would like to note that the above arguments about XML vs. RDF do not change substantially when regarding XML Schema instead of XML DTD's as the language in which to specify the structure of XML documents. Readers might be tempted to compare the "type extension" mechanism of XML Schema with the "subclassOf" mechanism of RDF Schema. However, the similarity between these two is only superficial. In fact, the "type extension" mechanism of XML Schema cannot at all be used to model ontological subtypes: in XML Schema, if type T' is derived from type T, then elements of the derived type T' are not necessarily members of the original type T. This is in contrast with the "subclassOf" relationship from RDF Schema, where any member of a subclass is by definition also a member of the original super-class. As a result, "subclassOf" can be used to model ontological subtyping, whereas XML Schema's type extension cannot.

## 5 Conclusion

In this paper I introduce the formalized knowledge is a crucial issue for the Web applications, and that such formalized knowledge can be represented by the current markup language, such as XML and RDF. The RDF data-model is sound and makes approaches from AI and Knowledge Engineering for establishing semantic interoperability directly applicable. Furthermore, it is universally extensible. This paper has shown the feasibility of applying markup language to a particular ontological modeling language, and the similar strategy could apply to any knowledge modeling language.

The arguments in this paper can be viewed from two different aspects. From the Web aspect, it is currently regarding XML as the most important step towards semantic integration. But for the development in the long run, RDF is a much better platform for this. For the agent aspect, XML and RDF can be used for Knowledge Representation of agent. Agents can communicate via XML/RDF message.

## References

- [1] Harold Boley et al. Tutorial on knowledge markup techniques, August 2000. Slides on ECAI 2000, Berlin.
- [2] Howard Smith et al. Share the ontology in xml-based trading architectures. *Communication of the ACM. March 1999, 1999.*
- [3] Robert J. Glushko et al. An xml framework for agent-based e-commerce. *Communication of the ACM. March 1999, 1999.*
- [4] Stefan Decker et al. The semantic web - on the roles of xml and rdf. *IEEE Internet Computing. September/October 2000, 2000.*
- [5] ETHOS. The extensible markup language (xml). <http://www.sgml.unet.com/xml.htm>. ETHOS Technology Briefings Series 1: Developments Shaping Internets and Intranets.
- [6] Nicola Guarino. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human and Computer Studies, special issue on The Role of Formal Ontology in the Information Technology, 1995.*
- [7] Rlaph Swich (eds.) O.Lassila. Resource description framework (rdf) model and syntax specification. Technical report, W3C Recommendation, 1999. <http://www.w3.org/TR/REC-rdf-syntax>.
- [8] J. Paoli T. Bray and C.M.Sperberg-McQueen (eds.). Extensible markup language(xml) 1.0. Technical report, W3C Recommendation, 1998.