AFRL-IF-RS-TR-2006-319
Final Technical Report
November 2006

INFOSPHERE CONCEPT EXPLORATION AND DEVELOPMENT (ICED)

ITT Industries, Inc.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-IF-RS-TR-2006-319 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
GENNADY R. STASKEVICH	JAMES W. CUSACK, Chief
Work Unit Manager	Information Systems Division
	Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* NOV 2006	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE INFOSPHERE CONCEPT EXPLORATION AND DEVELOPMENT (ICED)	**5a. CONTRACT NUMBER** FA8750-05-C-0271
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62702F
6. AUTHOR(S) Michael Maciolek	**5d. PROJECT NUMBER** ICED
	5e. TASK NUMBER 05
	5f. WORK UNIT NUMBER 03
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ITT Corporation, 2560 Huntington Ave, Alexandria VA 22303-1404	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFSE, 525 Brooks Rd, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-319

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 06-743

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Infosphere Concept Exploration and Development (ICED) project, conducted by ITT Corporation under contract to AFRL Information Directorate, provides concepts, methods, and a prototype software system presenting a Community of Interest (COI) infosphere with a consistent vocabulary definition capability. As information management systems become more widely used by COI, capabilities are increasingly needed to easily configure such systems to reflect COI needs and vocabularies, instead of those of a single predefined organization. High operation tempos demand equally responsive information systems that can be composed, dissolved and reconfigured to match the changing nature of the information Battlespace.

15. SUBJECT TERMS
Community of Interest, Dynamic, Infosphere, Net Centric, Operations Ready, Publish, Subscribe

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Gennady Staskevich
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	51	**19b. TELEPHONE NUMBER** (*Include area code)*

Table Of Contents

TABLES

FIGURES

Executive Summary

The Infosphere Concept Exploration and Development (ICED) project, conducted by ITT Corporation under contract to the Air Force Research Laboratory Information Directorate (AFRL/IF), provides concepts, methods, and a prototype software system presenting a Community Of Interest (COI) infosphere with a consistent vocabulary definition capability. As information management systems become more widely used by COI, capabilities are increasingly needed to easily configure such systems to reflect COI needs and vocabularies, instead of those of a single predefined organization. High operation tempos demand equally responsive information systems that can be composed, dissolved and reconfigured to match the changing nature of the information battlespace.

The result of this effort is a proof-of-concept technology enabling and supporting the instantiation, integration, and operation of information management tools within and across COIs. Results include:

- A prototype Type Management System (TMS) to support a COI information engineer in defining XSDs for use in COI information management
- An iterative and hierarchical schema development process supported by the TMS
- A scenario and a demonstration of the TMS operating under that scenario
- Directions identified for future research for the development of the TMS and the schema processes supported by the TMS.

Schemas developed and maintained with the TMS are represented in a hierarchy with four levels. Base Data Types, function as (non-user managed) atomic primitives. Elements combine a base Data Type and a description of each Element's meaning. Fragments represent discrete sets of COI information in terms of combinations of Elements and sub-Fragments. Fragments and Elements are combined into models; each Model can generate an XML Schema Definition (XSD). XSDs can be used to configure information management tools. The capability to define reusable Models, Fragments, and Elements allows the TMS user to specify information objects and their components processed within and between COIs.

The TMS provides two interfaces to users. The Graphical User Interface (GUI) to the TMS, an Eclipse plugin, supports the creation and management of Elements, Fragments, and Models. The GUI provides a concise visual display of the construction of schemas. The Web Services interface to TMS provides an automated capability to fetch XSDs created by the TMS. ITT envisions that other information management tools will use the Web Services in machine-to-machine interactions.

An overall goal of the project was to demonstrate the feasibility of a functional, capable tool supporting COI information engineers and abstracting computer science, XML, and XSD knowledge from the user, thereby enabling COI information engineers to focus on application area issues. ITT found that this goal can be attained if the tool functionality is limited, but, as capabilities are added to the tool, the user is increasingly likely to be required to know specific aspects of XML, XSD, and computer science.

The TMS and its demonstration show that COI information engineers configuring and reconfiguring information management tools can benefit from automated software tool support. The prototype support provided by the TMS can be improved by future research in expansion of TMS support to additional features of the XSD language, additional capabilities to the TMS GUI, additional capabilities of the TMS Web Services, additional TMS capabilities particularly in TMS data storage, and expansion of TMS usefulness in process support and in the environment in which the TMS operates. This report provides suggestions of future research directions in each of these five areas.

1.0 Introduction

This document is the final report for the Infosphere Concept Exploration and Development (ICED), a research project conducted by ITT Corporation, Advanced Engineering and Sciences Division, under contract to the Air Force Research Laboratory, Information Directorate (AFRL/IF).

1.1 Background

The Air Force Research Laboratory, Information Directorate (AFRL/IF) investigates, develops, maintains, and enhances Operational Information Management (OIM) capabilities. OIM is a Net-Centric information management program. Information Management concepts are being implemented in various software architectures and systems that provide supporting infrastructure. The DoD Metadata Registry (DDMR), Information Management Core Services (IMCS), and Net-Centric Enterprise Services (NCES) are examples of such Information Management infrastructure, of which the IMC Services is implemented under the OIM program. AFRL/IFSE continues to explore tools, concepts, and processes supporting the creation and management of stable and consistent infospaces for Communities of Interest (COIs).

A Community Of Interest (COI) is a group of users who collaborate on a set of tasks. The members of a COI may come from multiple organizations. For example, a COI with a military application might include members from multiple services or personnel from defense organizations under multiple governments. A COI needs a shared vocabulary to describe and share information among COI members and with other organizations. As Information Management systems become more widely used by COIs, capabilities are being increasingly demanded to easily configure such systems to reflect COI needs and vocabulary, instead of those of a single predefined organization.

The IMCS Metadata Schema Repository and Information Object Repository provides a capability in which XML Schema Descriptions (XSDs) are used to define information objects. The development of an XSD, however, requires fairly sophisticated technical abilities. Furthermore, it is difficult to browse XSDs and easily compare and contrast how different information objects are defined. Capabilities are needed to provide personnel that have specialized application knowledge, but less advanced computer science technical knowledge, methods and processes for configuring Information Management systems to satisfy COI needs.

Analysts designing information objects for a COI, and their associated schemas, draw on either an informal or formal representation of the domain from which information objects can come. This representation may include some notion of the individuals comprising the domain of discourses, the classes or sets into which these individuals fall, attributes that can describe individuals, and relations between individuals, classes, etc. In other words, the development of schemas for a COI falls within the discipline of ontological engineering. An ontology:

> "is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base." (Swartout et al. 1997, as quoted in Gomez-Perez et al. 2004)
>
> "An ontology may take a variety of forms, but it will necessarily include a vocabulary of terms and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms." (Uschold and Jasper 1999, as quoted in Gomez-Perez et al. 2004)

Researchers in Knowledge Engineering, Artificial Intelligence, and Computer Science increasingly find it useful to explicitly represent ontologies in a formal language.[1] For example, the Web Ontology Language (OWL, see http://www.w3.org/TR/owl-features/) is an extension of XML to support the specification of ontologies. Furthermore, researchers in these fields have developed methods, methodologies, and tools for constructing, merging, evaluating, and otherwise managing ontologies. For example, Protégé (http://protege.stanford.edu/) "is a free, open source ontology editor and knowledge-base framework". As another example, Jena (http://jena.sourceforge.net/ontology/) provides a Java-based Application Programming Interface (API) supporting the development of applications within the semantic web.

The OIM program has identified a need to provide ontological engineering capabilities to analysts setting up Information Management capabilities for a COI. Such analysts may be presumed to have a general technical background, but cannot be assumed to have completed training in XML, XSD, or tools and languages supporting ontological engineering. The research described in this report fills this need by demonstrating a software system and processes to provide these capabilities.

1.2 Purpose

The work described in this report provides concepts, methods, and a prototype software system presenting a COI infosphere with a consistent vocabulary definition capability. An overall goal of the project was to demonstrate the possibility of a functional, capable tool supporting COI information engineers and abstracting computer science, XML, and XSD knowledge from the user. ITT wanted to develop a user-friendly tool for COI staff without an XML expert available. Specifically, this effort:

- Developed a method for defining a COI's vocabulary using primitive terms:
  - The software system provides a Type Management System (TMS) in which base Data Types function as (non-user managed) atomic primitives

[1] Quine (1953) is an early case in philosophy of analyzing formal languages to clarify ontological commitments. Quine's slogan, "To be is to be the value of a bound variable" concisely states that the sets of objects which quantified variables can range over in statements of the predicate calculus, such as $\forall x, p(x)$, constitute those individuals whose existence is postulated by an analysis of statements.

    - The software system supports a process for creating a COI-specific vocabulary of Elements in the TMS, where an Element combines a base Data Type and a description of the Element's meaning
    - The software system, associated documentation, and associate processes provides capabilities to manage the COI-specific vocabulary
- Developed a process for defining schemas using the COI's defined vocabulary
    - The software system supports representing discrete sets of COI information in terms of combinations of primitive terms; these combinations are the Fragments in the TMS
    - The software system supports combining Fragments and Elements into Models, which in turn can be used to generate XML Schema Definitions (XSDs). These XSDs are compatible with the IMCS, as well as with other systems.
- Developed a software system enabling clients to define COI vocabulary and to construct XSDs in accordance with the COI's ontological requirements
    - The Graphical User Interface (GUI) to the TMS, an Eclipse plugin, supports the creation and management of the primitive Elements in a COI-specific vocabulary
    - The GUI allows for the visual display of the construction of schemas through the creation and management of Fragments and Models, comprised of Elements and Fragments
    - The Web Service (WS) interface to the software system provides the TMS with a capability to fetch XSDs generated by the COI, which can then be used in other systems, such as the IMCS Metadata Schema Repository (MSR)
- Demonstrated the newly developed capabilities for this effort, as encapsulated in the TMS
    - The demonstration, as exhibited at the effort's final presentation and described in Section 3.3 of this report, is defined in the context of an operationally realistic scenario
    - The setup, initialization, and execution of the demonstration are documented in this report.

1.3 Scope

As part of a grander vision of composable Infospaces, the scope of this effort is to design and develop methods, processes and software that provide a consistent vocabulary management capability to COI members and administrators.

1.4 Synopsis of Report Organization

The remainder of this report is subdivided into two main sections and followed by a conclusion.

Section 2, titled "Methods, Assumptions, and Procedures", describes the tasks conducted under this effort to achieve ITT's research results. Section 2 describes the development of methods for defining a COI-specific vocabulary, of processes for defining schemas, of software to provide automated support for these methods and processes, and of a demonstration of that tool. Section 2 also briefly describes the management of the ICED project.

Section 3, title "Results and Discussion", describes the research results achieved under this effort. Results fall into three main categories:

- The Type Management System (TMS)
- The specification of schema development processes supported by the TMS
- The definition of a scenario with which the TMS is demonstrated

Section 3 describes results in each of these areas. As part of the description of the TMS, instructions are provided for deploying and executing the software.

The conclusion section summarizes the accomplishments of this effort and directions for future research.

2.0 Methods, Assumptions, and Procedures

The methods and procedures ITT adopted under the ICED project fall into four tasks:

- Develop a method for defining a vocabulary for a Community Of Interest (COI)
- Develop a process for defining schemas for information objects
- Develop the Type Management System (TMS)
- Develop a demonstration of the TMS

This section describes the activities conducted under each one of these tasks. This section also briefly overviews aspects of ICED project management.

2.1 Develop Method for Defining a Vocabulary

ITT investigated how a vocabulary for a COI would be used. In the context of information management, the terms in a COI vocabulary are the names of information objects and parts of information objects. In particular, a vocabulary would be used to describe information objects in terms of metadata schemas. Schemas and parts of schemas support searches (Harlow 2005) and the population of Metadata Registries (ISO/IEC 2004), such as the DoD Metadata Registry (DDMR), and of the Metadata Schema Repository (MSR) associated with the Information Management Core Services (IMCS).

2.1.1 Create terminology

Under the ICED project, a hierarchical structure was created in which the vocabulary for a COI can be formalized. Terminology was invented to describe terms on each level of this hierarchy:

- Data Types, originally called "Atomic Primitives", are the lowest level in the hierarchy and are predefined
- Elements, originally called "Complex Primitives", are basic building blocks of a COI vocabulary
- Fragments are composed of Elements and other Fragments.
- Models, originally called "Schemas", are the highest level of the hierarchy, are composed of Elements and Fragments, and are used to generate XML Schema Definitions (XSDs).

2.1.2. Adopt base set of Data Types

The base set of Data Types were selected from the data types defined in XSD. These types were selected to illustrate how more complex notions can be built on top of this base in defining a COI vocabulary.

2.1.3 Develop a method for creating a COI-specific vocabulary

Once the hierarchy of terms was defined, one can explore how a COI vocabulary can be created and structured in this hierarchy. In particular, the use of subfragments, as children of Fragments, allows any level of depth to be supported by this hierarchical method. Tool support is easier to provide if the hierarchical method is confined to producing a Directed Acyclic Graph (DAG), and that constraint was imposed on the prototype produced under this project.

Section 3.1.1 provides further details about Data Types, Elements, Fragments, and Models.

2.2 Develop a Process for Defining Information Object Type Schemas

In specifying a process for defining schemas, ITT researchers drew on the literature in ontological engineering and in software engineering lifecycle models. Gomez-Perez et al. (2004) discusses how ontology development processes fit into a larger set of processes, including management and support processes. The schema processes were defined in an analogous context. Furthermore, schema development was conceptualized as a set of iterative processes. Iterative or spiral lifecycle processes have been used in software engineering for some time (Boehm 1988). Top-down and bottom-up processes are well-established metaphors in software engineering, and these metaphors were drawn on for further specifying schema development processes. Schema processes, including schema development processes, are more fully defined in Section 3.2

2.3 Develop Type Management System

ITT developed a software system, the Type Management System (TMS), under the ICED project. ITT met with AFRL during the ICED kickoff meeting to clarify the role and purpose of the TMS. This clarification, and the definition of terms and individual pieces of the TMS, solidified the concepts and high-level architecture of the TMS.

In particular, ITT and AFRL agreed that the TMS is a repository for Elements, Fragments, and Models that represent information relevant to Communities of Interest (COI). The TMS has a Graphical User Interface (GUI) that allows users to create, modify, and delete that information. The TMS presents appropriate services to the community via a Web Service layer. With this capability, the TMS could serve as the Metadata Schema Repository in a Net-Centric Information Management System, such as Information Management Core Services (IMCS) implementations. ITT and AFRL also agreed that a secondary goal of the TMS is to provide enough ontological support so that users can convey the meaning of their COI’s information.

ITT investigated the Web Ontology Language (OWL), Protégé, and Jena, which are technologies that can be leveraged in developing ontology tools. Both Protégé and Jena can be used inside Eclipse. ITT installed and experimented with both using simple models. Since TMS storage was designed as XML flat files, use of Protégé and Jena would be overkill for the ICED project. These advanced technologies may be used in future work adding advanced ontology capabilities to the TMS. For example, Jena could be used with the TMS model if capabilities to manipulate OWL schemas are desired.

This experimentation helped establish that the TMS development would focus on initial storage capabilities, a Web Services interface, and a GUI.

2.3.1 TMS Storage

The TMS allows the user to define Elements, Fragments, and Models. The TMS generates XSD files. The Elements, Fragments, and Models must be stored in an intermediate form to support user definitions and management. ITT researched repository solutions and implemented a prototype TMS storage solution.

For example, ITT looked at the Berkeley XML Database (XMLDB). After research on the stability and maturity of XMLDB, ITT decided that this project would not benefit from implementing a solution based on an XMLDB. ITT also looked at using Jena as the database interface to a MySQL Database.

ITT decided that, for initial capabilities produced under this project, TMS storage would use simple file system storage mechanisms. ITT did not implement a solution using a full relational or XML database. This decision sped up development and allowed ITT to focus on prototype functionality instead of working on underlying database infrastructure. The TMS storage capabilities are further described in Section 3.1.2.

2.3.2 Web Services Development

Under the direction of AFRL, the development of the TMS was decoupled from IMCS dependencies. The TMS provides a capability for information management tools (IMCS, NCES Discovery Services, etc.) supporting COIs to retrieve XML schemas created with the TMS. That is, the Web Services interface to the TMS provides Web Services to access schemas generated by the TMS.

ITT began Web Services development by implementing skeletal Web Services. ITT expanded and refined these over the course of the project. ITT explored some Web Services capabilities that could be implemented in further work. For example, ITT researched:

- The use of Simple API for XML (SAX) and Document Object Model (DOM) libraries and constructed preliminary methods to provide a capability to search the contents of XSD files
- The construction of a demonstration client for a Web-based front end to TMS, providing schema-based searching on either a Java console application or as a Web application using the same Web Services. This research included experimentation with Ruby on Rails.

At the completion of the project, the TMS Web Services provided capabilities to:

- Retrieve XSDs by name
- Retrieve a list of XSDs in a specified directory
- Search for names of XSDs based on a partial name

The TMS Web Services are more fully described in Section 3.1.4.

2.3.3 GUI Development

At the ICED kickoff meeting, ITT described plans to implement the TMS GUI in the Eclipse framework. In keeping with these plans, ITT programmers explored the Standard Widget Toolkit (SWT) - which is a Java-based GUI development framework used in Eclipse, Eclipse application development, and the Eclipse plug-in architecture. The TMS GUI displays Elements, Fragments, and Models. (Originally, “Models” were called “Schemas”.) Properties, corresponding to XSD “Attributes”, can be added to Fragments and Models. ITT used the GUI development as the driver for defining the underlying logic for maintaining Elements, Fragments, and Models.

In February 2006, ITT conducted a Technical Interchange Meeting (TIM) with AFRL in which various project decisions were discussed and debated. It was at this meeting that the distinction between Fragments and Models was introduced into the TMS design. Another topic of discussion was how changes made in the GUI would affect existing XSDs. That is, matching XSDs would be updated when an Element, Fragment, or Model is changed through the GUI.

The development of the TMS GUI continued to progress after the TIM. Capabilities added to the GUI include:

- The generation of XSD from a Model and the saving of the XSD to a file
- The updating of any matching XSD when an Element, Fragment, or Model is changed
- Some drag-and-drop functionality – dragging an Element to a Fragment or Model, dragging a Fragment under another Fragment, and dragging a Fragment to a Model now all modify the hierarchy of user-defined Elements, Fragments, and Models appropriately
- A “View XSD” feature
- The listing of the Fragments and Models in which a user-specified Element is used
- The sorting of Elements
- The addition of Properties to Fragments and Models
- A robust “Help” feature, including the addition of many Help topics
- The capture of the time and user when Elements, Fragments, or Models are updated

- Finer-grained logging to record user modifications

Developing the TMS GUI included many details necessary to produce a working system. For example, the code constrains the decomposition of Fragments to be a loop-free tree, thereby avoiding infinite loops in the code. Some effort was expended towards the end of the project in researching how to deploy the GUI as an Eclipse plugin. The code was documented and Javadocs were generated for all TMS classes. ITT conducted much informal testing throughout the development process. A number of known schemas were generated with the tool as part of the testing. The TMS GUI is more fully described in Section 3.1.3.

2.4 Demonstrate the TMS

The TMS is a proof-of-concept system intended to show the utility of providing COI information engineers with capabilities to model the structure of COI information objects and to generate schemas. As such, developing a demonstration of the use of the TMS was an important part of the ICED project.

ITT presented two demonstrations of the TMS during this project. The first was a presentation at the Operational Information Management (OIM) Principal Investigator (PI) meeting in April 2006. The TMS software was deployed for this presentation to run off a flash drive. The focus of this demonstration was to illustrate some capabilities of the TMS, as it stood at that stage of the project.

ITT demonstrated the TMS as part of the ICED final presentation in September 2006. This demonstration was presented in the context of a user-oriented scenario, in which a COI information engineer uses the TMS to create a Cursor-On-Target (COT) XSD file to configure COI information management tools. As such, the scenario and demonstration provide a context in which schema processes can be discussed. The demonstration scenario and TMS demonstration is more fully described in Section 3.3.

2.5 Project Management

Project management activities performed by ITT during this task included status reporting and meetings with AFRL/IF. Table 2-1 lists major meetings with AFRL/IF participation.

Table 2-1: ICED Meetings

Review	Date
Kickoff Meeting	17 October 2005
Technical Interchange Meeting (TIM)	3 February 2006
Presentation at OIM PI Meeting	11-12 April 2006
Status Review Meeting	23 May 2006
Status Review Meeting	15 June 2006
Final Presentation	15 September 2006

At the kickoff meeting, ITT assisted in defining the requirements for the TMS. A discussion of current research on ontologies and AFRL/IF feedback on the tasks in the Statement of Work (SOW) helped clarify the scope of the ICED project. ITT and AFRL/IF agreed that basic terms of the TMS include “Data Type”, “Element”, and “Fragment”.

ITT prepared and presented slides at the February TIM. ITT discussed project progress and reported no obstacles to progress existed in the ICED project.

ITT personnel participated in the OIM PI meeting held April 11-12 in Washington, DC. ITT presented a poster session, a demonstration, and a slide show.

ITT met with AFRL/IF on 23 May to discuss the tasks to be completed before the end of the contract. ITT and AFRL/IF agreed that they would hold another meeting to discuss the priority of enhancements extending the ICED scope if the tasks identified in the SOW were completed without depleting the funding.

This additional meeting was held on 15 June. ITT and AFRL/IF discussed the project tasks as outlined in the statement of work. For those tasks that were incomplete, ITT and AFRL/IF agreed on a path to completion.

At the final presentation, ITT presented the results of this effort. Results included a description of the benefits of using the TMS, of directions for future research, of schema definition processes, and of the scenario for the demonstration. The TMS was demonstrated at the final presentation.

3.0 Results and Discussion

Under the ICED project, ITT:

- Produced a prototype Type Management System (TMS) to support a Community Of Interest (COI) information engineer defining information objects to be used in COI information management
- Identified schema processed and defined schema development processes supported by the TMS
- Defined a scenario and produced a demonstration of the TMS operating under that scenario.

Each one of these results is discussed in a subsection of this section.

3.1 Type Management System

The Type Management System (TMS) is comprised of three major subsystems:

- XML files for storing Elements, Fragments, Models, and Schemas manipulated by the TMS
- A Graphical User Interface (GUI), deployed as an Eclipse plugin
- TMS Web Services for accessing and retrieving XML Schemas created by the TMS.

After an overview of the TMS, each major subsystem is described in a subsection of this section.

3.1.1 Type Management System Overview and Vocabulary

The TMS is designed to support a user defining, modifying, or configuring schemas for Information Management tools. These Information Management tools are assumed to be capable of providing infrastructure for a Community Of Interest (COI). The TMS provides capabilities for describing information objects that flow between and within such COI tools. In effect, the TMS allows the user to build and maintain a vocabulary for a COI, where the words in the vocabulary can range over the names of these information objects and of parts of these objects.

The TMS GUI provides a user with a capability to create and browse a compact description of potentially complex XML Schema Definitions (XSDs). Files containing XSDs can be generated from Models in the TMS, while Fragments and Elements provide reusable components of Models. These XSD files, in turn, can be used to configure Information Management tools, such as IMCS and NCES servers and clients.

The TMS user is assumed to be a COI information user. The user is assumed to have domain knowledge, where the domain would vary with the specific COI and the applications supported within that COI. For example, in the scenario developed under this project to demonstrate TMS capabilities, the user is assumed to have some knowledge of tactical military applications. The backend of the TMS draws heavily on XML and XSD standards and technology. The TMS user, however, need not understand either XML or XSD syntax. The TMS user is assumed to have some introductory computer science knowledge. Specifically, the user is assumed to be aware that variables are typed; to recognize type names, such as float and integer; and to be able to decide what types are appropriate for Elements and Properties the user defines within the TMS.

The Type Management System (TMS) is organized around a hierarchy of user-defined Elements, Fragments, and Models. Elements are defined in terms of given Data Types, and XML Schema Definitions (XSDs) are generated from Models. This overview of the TMS explains what Data Types, Elements, Fragments, Models, and Schemas are.

3.1.1.1 Data Type

A Data Type, in the TMS, is a low-level basic component of a COI vocabulary. Data Types have no semantics associated with them, other than some general mathematical and calendar knowledge. The TMS currently has the following Data Types defined: string, integer, nonNegativeInteger, positiveInteger, decimal, float, date, time, dateTime. These are a subset of Data Types defined in standard XSD. Data Types are used by TMS users creating Elements and assigning Properties to Fragments or Models, but are not explicitly managed by TMS users.

3.1.1.2 Element

An Element is a higher-level object than a Data Type. Elements are the basic building blocks of the TMS, representing the smallest or atomic unit of information managed by the TMS user. An Element consists of a Name, Data Type, and Description. These three fields are required. The TMS also contains information on when and by whom each Element was created and last modified. With the mandatory fields in an Element, Elements describe a data item with semantic information not present in Data Types (Figure 3-1).

An Element’s Name must be unique across the TMS. No other Element, Fragment, or Model can have the same Name. Names should be chosen to be meaningful to others within the COI. The Element’s Description should explain the Element’s purpose and use in enough detail to be understandable to others in the COI. The reuse of Elements, Fragments, and Models is a goal of the TMS, and the Description of an Element should support such reuse.

3.1.1.3 Fragment

Fragments are snippets of the COI-specific language that is being defined. Fragments are composed of Elements and other Fragments. Fragments differ from Models in that Fragments are incomplete, whereas Models are a finished product. Models can be used to generate XML Schemas, but Fragments cannot.

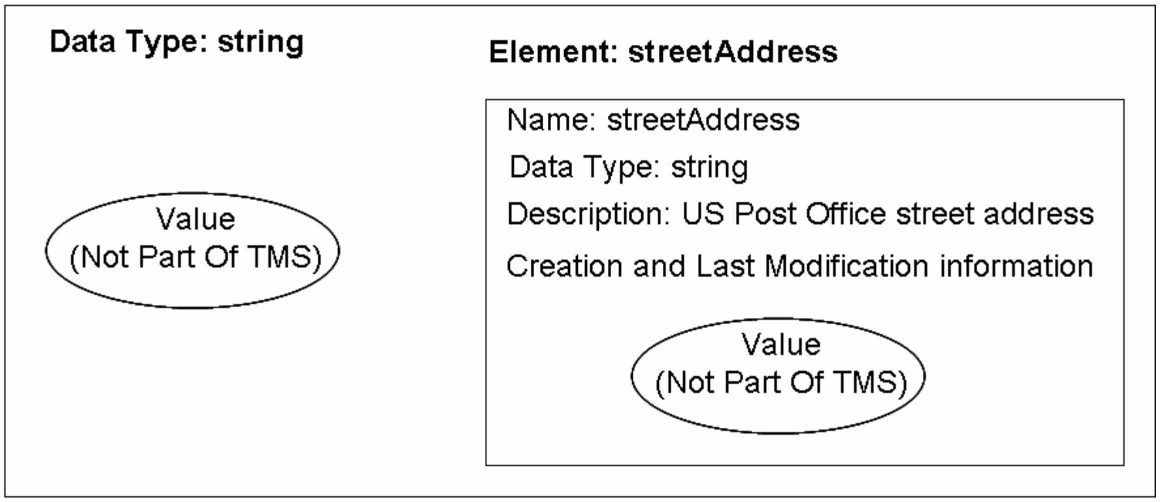


Figure 3-1: Data Types Contrasted With Elements

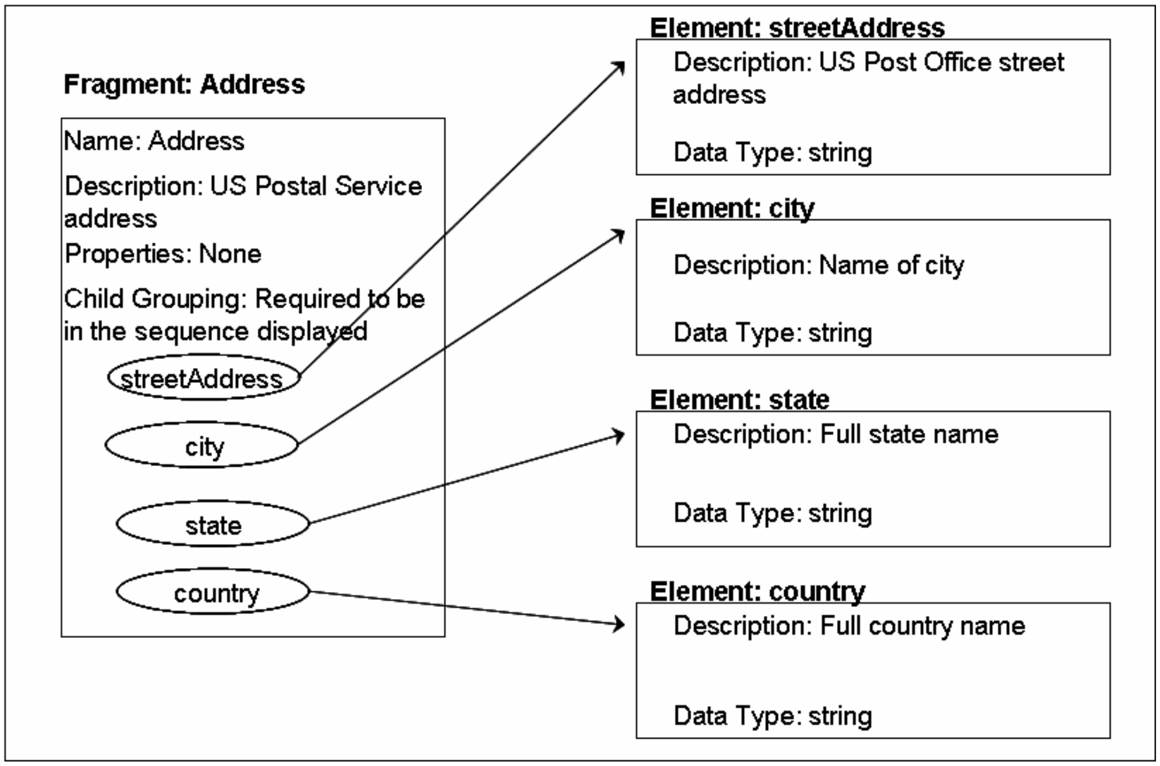

Fragment: Address
Name: Address
Description: US Postal Service address
Properties: None
Child Grouping: Required to be in the sequence displayed
streetAddress
city
state
country
Element: streetAddress
Description: US Post Office street address
Data Type: string
Element: city
Description: Name of city
Data Type: string
Element: state
Description: Full state name
Data Type: string
Element: country
Description: Full country name
Data Type: string

Figure 3-2: A Fragment with Elements as Children

A Fragment consists of a Name, Description, Properties, Child Grouping, and Children. Name and Description are mandatory fields, and the TMS assigns a default Child Grouping to each Fragment. A Child can be an Element or another Fragment. Properties and Children are optional; a Fragment can have zero Children and no Properties. When a Fragment has more than one Child, some of those Children can be Elements and some can be Subfragments. No restriction is imposed that all Children must be either Elements or Subfragments; Children can be mixed.

The Fragment's Name must be unique across the TMS. No Element, other Fragment, or Model can have the same Name. As with Elements, Names and Descriptions should be defined to be meaningful and to support reuse.

Additional information about the Fragment can be added as a Fragment Property. Properties, unlike Elements, are not reusable in other Fragments. A Fragment Property corresponds to an attribute in XML.

Fragments and Subfragments can only form loop-free trees in the TMS. In other words, a Fragment cannot be a Child of itself, and A Fragment cannot be simultaneously a descendant of another Fragment and also an ancestor of that Fragment.

The Child Grouping determines the order and number of occurrences of the Children. If there are no Children, the Grouping is ignored. Three Child Grouping choices exist:

- Required to be in the sequence shown: each Child can appear any number of times, but they must be in the order in which they appear in the TMS GUI
- Unordered (but Children can appear at most one time): Any of the Children can be omitted. Each Child that is used can appear at most once. The Children that appear can be in any order.
- Mutually exclusive (only one Child can appear): Only one can appear, but that type of Child can appear any number of times.

These Child Grouping options correspond to the three varieties of model groups in XML Schema Definitions: sequence, conjunction ("all"), and disjunction ("choice"). In the XSD specification, "sequence", "all", and "choice" are the three kinds of compositors.

3.1.1.4 Model

An information model is a pattern describing an information object. The model defines the information object's parts, properties, and the relationships between the different parts. A TMS Model consists of a Name, Description, Properties, Child Grouping, and Children. Name and Descriptions are mandatory fields, and the TMS assigns a default Child Grouping to each Model. A Child can be an Element or a Fragment. Properties and Children are optional; a Model can have zero Children and no Properties. When a Model has more than one Child, some of those Children can be Elements and some can be Fragments. No restriction is imposed that all Children must be either Elements or Fragments; Children can be mixed.

Other characteristics of a Model also parallel the characteristics of a Fragment. The Model's Name must be unique across the TMS. No Element, Fragment, or other Model can have the same Name. As with Elements and Fragments, Names and Descriptions should be defined to be meaningful and to support reuse. Additional information about the Model can be added as a Model Property. Properties, unlike Elements and Fragments, are not reusable in other Models. The Child Grouping of a Model has the same options and meaning as the Child Grouping for a Fragment.

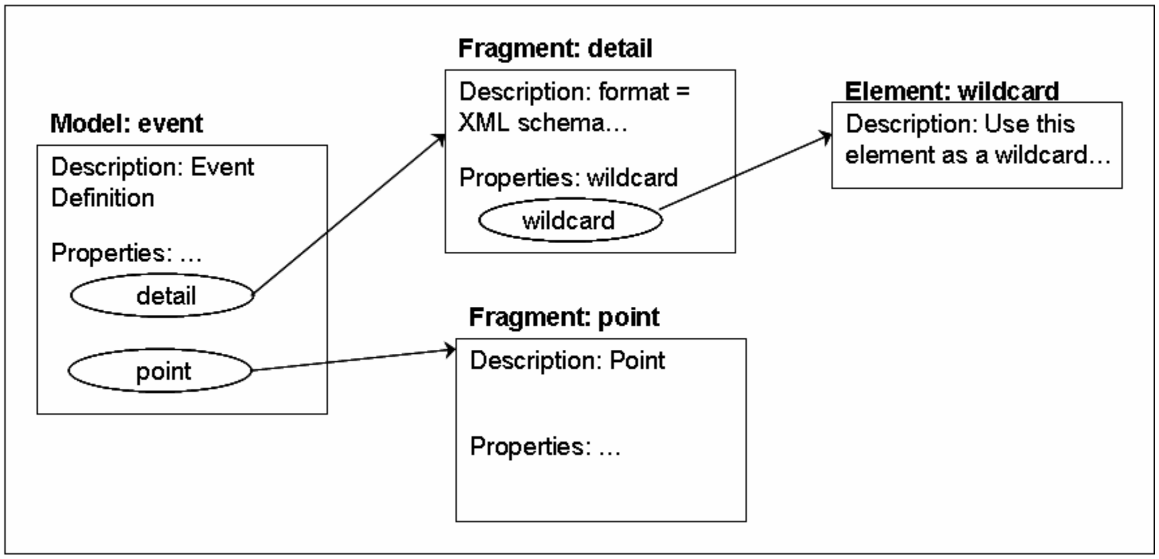


Figure 3-3: A Model for Cursor On Target

3.1.1.5 Schema

An XML Schema Definition (XSD) can be generated and saved from each Model. The XSD is written to a file with the name *modelName.xsd* where *modelName* is the unique Name of the Model. Generated schemas can be accessed via function calls in the TMS Web Service layer.

3.1.2 Type Management System Storage

The TMS stores Elements, Fragments, Models, and Schemas in files. In addition, log files include messages about modifications made with the Graphical User Interface (GUI) to TMS data. The locations of these files are specified in a "properties" file deployed with the GUI. The TMS Web Services deployment specifies the directory containing the XML Schema Definition (XSD) files generated by the TMS.

Information on Elements, Fragments, and Models is stored in XML files. The formats of files for Elements, Fragments, and Models are described in the GUI on-line help system. As described in the on-line help, each XSD file contains a valid XML Schema definition. A Schema can be generated for each Model, and these Schemas can be used with other information management applications.

The simplicity of the TMS storage system reflects a design decision to concentrate on novel capabilities. The TMS does not currently include versioning, but some protections are built into the GUI for the user to recover Elements, Fragments, and Models from undesired changes. When modifications are made using the GUI, the changes are made for the current session only until the changes are saved. If changes have been made since the last save that the user would like to undo, exiting the GUI without saving will leave Elements, Fragments, and Models as defined in the last save.

The Elements, Fragments, Models, and XSDs are stored in specific directories as identified in the properties file deployed with the GUI. All Elements are stored in one file. Each Fragment, Model, and XSD is stored in a separate named file. Therefore, one can make a backup copy of the TMS data before saving changes. Using the file system to backup and restore TMS files is not without risk. If a restored file is out of sync with other files, the item will not be loaded when the TMS GUI is next launched. For example, suppose some Elements are deleted and a Fragment is changed in a TMS GUI session to no longer refer to the deleted Elements. If a copy of the Fragment file is restored to the state prior to the change, an inconsistency will arise. Since the Elements file was not saved and restored with the Fragment, Elements in the restored Fragment no longer exist, and, therefore the Fragment cannot be loaded.

These sort of inconsistencies cannot arise with XSDs. XSD files are complete, valid, well-formed XML schemas. Before generating and saving an XSD file for a modified Model, you may want to consider renaming the last previously-generated XSD file for that Model. When the user employs the TMS GUI to generate and save an XSD file for a Model, the GUI checks to see if an XSD file already exists for that Model. If an XSD file exists, a message is displayed asking the user if the existing XSD file should be copied over. The XSD file can be renamed, by using the file system outside the TMS GUI, at this point.

3.1.3 Graphical User Interface to Type Management System

The TMS includes a Graphical User Interface (GUI). The GUI provides capabilities to support a COI analyst in defining a COI vocabulary, in combining Elements of that vocabulary, and ultimately in generating Schemas describing information objects needed to support information management within and between COIs.

The TMS GUI is an Eclipse plugin and was developed under the Eclipse Integrated Development Environment (IDE). According to their website http://www.eclipse.org,

> “Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software.”

One can purchase various books (for example, Burnette (2005) or Gallardo et al. (2003)) describing how to use Eclipse to develop software.

3.1.3.1 GUI Software

The TMS GUI software is distributed under this project in two zip files. One contains source and development files. The other contains the binaries that are deployed as an Eclipse plugin. The software and development files consist of:

- Various settings and configuration files that organize the software as an Eclipse project
- bin: folder of compiled Java classes for the TMS GUI
- doc: folder containing Javadoc files for the TMS GUI
- html: folder containing html files to incorporate in Eclipse help for the TMS GUI
- icons: folder containing an icon for the TMS GUI
- META-INF: folder containing a manifest file
- src: Java source for the TMS GUI

The source code is organized as one Java package, consisting of the Java classes shown in Table 3-1. The TMS GUI is compiled, and the ICED jar file is created, from within Eclipse.

Table 3-1: Java Source for the TMS GUI

Pkg.	**Name**	**Type**	**Notes**
iced	AddToFragmentDialog	class	extends org.eclipse.jface.dialogs.Dialog;
	Attribute	class	
	AttributeDialog	class	extends org.eclipse.jface.dialogs.Dialog
	ChildElement	class	
	ChildElementDialog	class	extends org.eclipse.jface.dialogs.Dialog
	ChildFragment	class	
	ChildFragmentDialog	class	extends org.eclipse.jface.dialogs.Dialog
	Constants	class	
	Element	class	implements java.lang.Comparable
	ElementContentProvider	class	implements org.eclipse.jface.viewers. IStructuredContentProvider extends org.xml.sax.helpers.DefaultHandler
	ElementDialog	class	extends org.eclipse.jface.dialogs.Dialog
	Fragment	class	
	FragmentContentProvider	class	implements org.eclipse.jface.viewers. ITreeContentProvider
	FragmentDialog	class	extends org.eclipse.jface.dialogs.Dialog
	FragmentXMLParser	class	extends org.xml.sax.helpers.DefaultHandler
	ICEDPlugin	class	extends org.eclipse.ui.plugin. AbstractUIPlugin
	ICEDView	class	extends org.eclipse.ui.part.ViewPart
	StringArrayTransfer	class	extends org.eclipse.swt.dnd. ByteArrayTransfer

3.1.3.2 GUI Deployment

The GUI to TMS is deployed as an Eclipse plug-in. ITT has tested it under Eclipse versions 3.1.0 and 3.2 on a Windows platform. It has also been run under Eclipse version 3.2 on under Mac OS X, version 10.4.5, although these instructions do not quite apply to a Macintosh platform. To install the GUI, move the iced_1.0.0.jar file and the iced folder to your eclipse\plugins directory. The deployment consists of the following files:

- eclipse\plugins\iced_1.0.0.jar
- eclipse\plugings\iced\tms.properties
- eclipse\plugings\iced\elements\elements.xml
- eclipse\plugings\iced\fragments\README.txt
- eclipse\plugings\iced\models\README.txt

One can delete the files named “README.txt”. These files are placeholders created to ensure that the directories in which they reside are carried along in the deployment. The GUI is deployed with a default “wildcard” Element and no Fragments, Models, or Schemas. To deploy example Elements, Fragments, and Models, see the scenario and demonstration described in Section 3.3. The user can redefine the location in which the GUI stores TMS data by modifying the tms.properties file (Figure 3-4).

```
// Indicates a comment line

// The directories are relative to the directory from which Eclipse is running, unless
// the complete path is included

element.directory=plugins/iced/elements/
element.filename=elements.xml
fragment.directory=plugins/iced/fragments/
model.directory=plugins/iced/models/
xsd.directory=plugins/iced/XSDs/
log.directory=plugins/iced/logs/
```

Figure 3-4: tms.properties File Specifies TMS Storage Locations

3.1.3.3 GUI Execution

The GUI to the TMS is launched from within the Eclipse SDK. Eclipse asks the user to specify a workspace when Eclipse is being started. Choose any workspace you like here; this choice is irrelevant to the TMS GUI plugin. Eclipse presents a series of pull-down menus (File, Edit, Navigate, Search, Project, Run, Window, Help) at the top of the Eclipse window. Choose the Window menu, the “Show View” option on the pull down menu that appears, and then “Other” on the final submenu. A pop-up window will then appear, from which the user should choose “COI TMS” under the “COI TMS” folder. The user may want to maximize the COI TMS view. Figure 3-5 illustrates the results of these steps launching the TMS GUI.

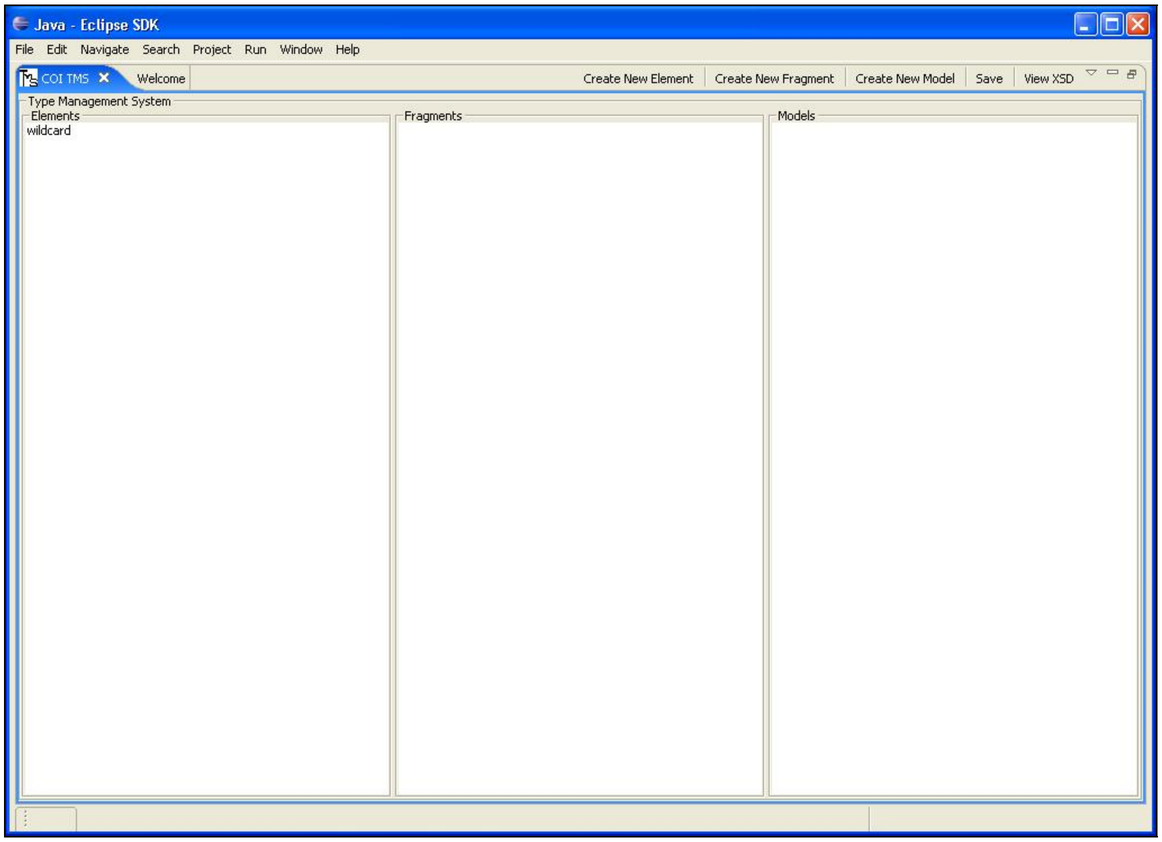


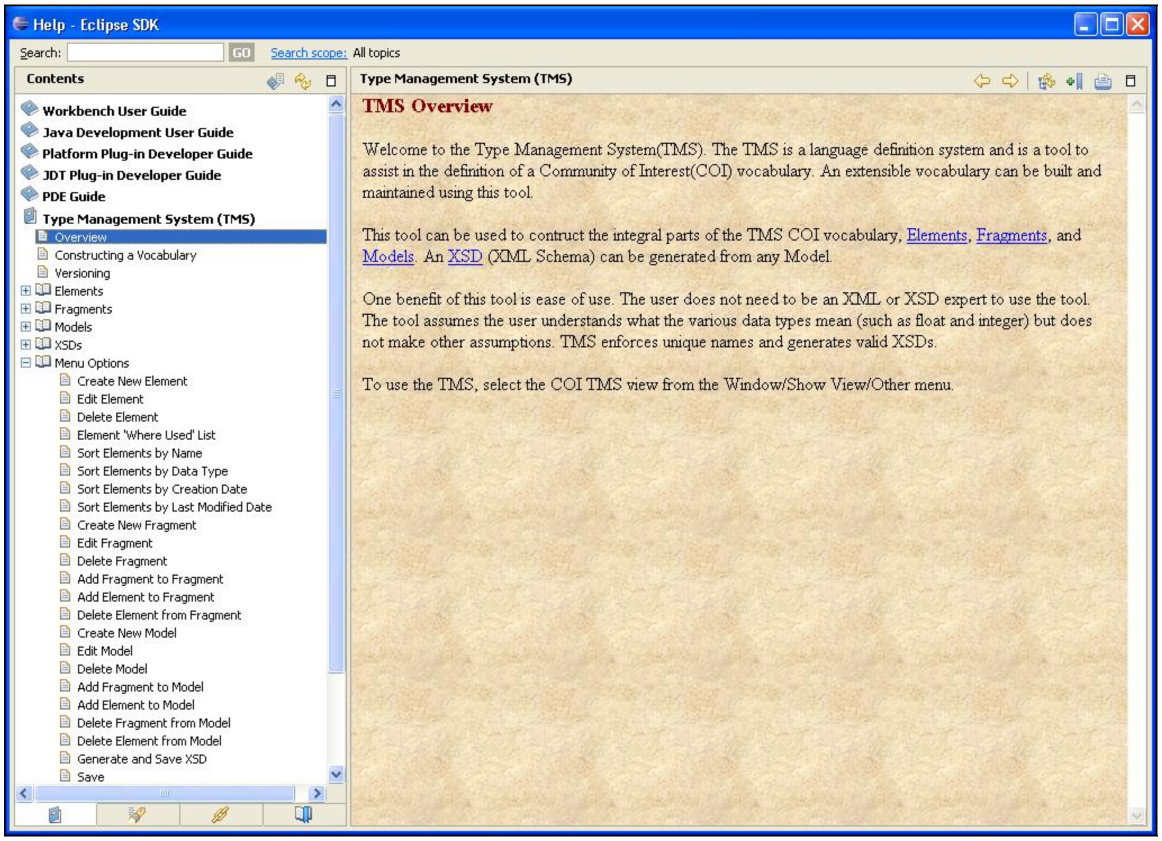
Figure 3-5: ICED View Launched As An Eclipse Plugin

Once the TMS GUI has been launched, the user can create, modify, and save Elements, Fragments, and Models, thereby defining a COI vocabulary. From Models, the user can generate and save XSD files for describing COI information objects. The TMS GUI includes an extensive on-line help system describing these operations. To launch TMS GUI help, choose the Eclipse Help pull down menu and the "Help Contents" option on that menu. Figure 3-6 illustrates Eclipse help, with the TMS help expanded on the left panel.

3.1.4 Web Services Interface to Type Management System

The TMS includes a Web Services interface. The TMS Web Services perform simple processes. These Web Services support an interaction between an open standard input and the wide ranging functionality in the TMS. The open standard input is a Simple Object Access Protocol (SOAP) message delivered over TCP/IP. The Web Services focus on the retrieval and searching of Schemas generated by the TMS. The TMS Web Services consist of:

- getSchema
- getSchemaList
- getSchemaListfromPartialName



TMS Overview

Welcome to the Type Management System(TMS). The TMS is a language definition system and is a tool to assist in the definition of a Community of Interest(COI) vocabulary. An extensible vocabulary can be built and maintained using this tool.

This tool can be used to contruct the integral parts of the TMS COI vocabulary, Elements, Fragments, and Models. An XSD (XML Schema) can be generated from any Model.

One benefit of this tool is ease of use. The user does not need to be an XML or XSD expert to use the tool. The tool assumes the user understands what the various data types mean (such as float and integer) but does not make other assumptions. TMS enforces unique names and generates valid XSDs.

To use the TMS, select the COI TMS view from the Window/Show View/Other menu.

Figure 3-6: TMS Help Accessible Through GUI

The getSchema Web Service contains only one parameter, the name of the Schema to be obtained. The name (for example, Genes.xsd) references the Schema on the server side. This Web Service returns the Schema to the Web Service Client.

The getSchemaList requires no parameters and simply returns a list of all of the Schemas available for searching or retrieval.

The getSchemaListfromPartialName Web Service uses only a single string parameter, representing a partial match of Schemas to for which to search. For example, the string “GE” or “ge” returns all Schemas with names containing textual subsets which match. This Web Service treats its parameter as case insensitive. The string “GE” could return Schemas named “Genes.xsd” and “Page.xsd”. A Web Service client can use each Schema name returned from getSchemaListfromPartialName as the parameter to the getSchema Web Service.

3.1.4.1 TMS Web Service Software

The software implementing the TMS Web Services consists of two parts: software to install on the Web Services server and Web Services client software. The TMS server must have installed Apache Tomcat 5.x (http://tomcat.apache.org/), an open source application server. Both the server and the client must have Java 1.5 installed. Various open source libraries are automatically installed as part of the deployment process. Look in TMS/WEB-INF/lib after the server-side Web Services is deployed, or in TMSClient.jar in the client deployment to see these library packages.

The TMS Web Services software delivered under this project consist of three archives:

- TMS-WS.zip – Files to deploy under Tomcat on the TMS Web Services server
- Type Management System.zip – TMS Web Services source code and related development files
- TMSClient.jar – Default TMS Web Services client.

The source code for the TMS Web Services is organized into two packages (Table 3-2). The Java interface and classes in the package com.itt.tms.ws, along with an XML file, WSDL file, jar files, etc., provide TMS Web Server capabilities to be deployed on a server. The interfaces and classes in the package tms.client, along with jar files for certain open source tools, provide a default TMS Web Services client. The TMS Web Services development files include an Ant script. This Ant script automatically builds and deploys the TMS Web Services. The script depends on application settings (JAVA_HOME, CATALINA_HOME, etc.) defined at its initialization.

Table 3-2: Organization Of Java Source Code In TMS Web Services

Package	**Name**	**Type**	**Notes**
com.itt.tms.ws	ISchemaService	interface	extends java.rmi.Remote
	SchemaNotFoundException	class	extends java.lang.Exception
	SchemaService	class	implements ISchemaService, javax.xml.rpc.server.serviceLifecycle
tms.client	ISchemaService	interface	extends java.rmi.Remote
	ISchemaServicePortSoap\ BindingStub	class	extends org.apache.axis.client.Stub, implements ISchemaService
	SchemaNotFoundException	class	extends org.apache.axis.AxisFault, implements java.io.Serializable
	SchemaService	interface	extends javax.xml.rpc.Service
	SchemaServiceClient	class	extends javax.swing.JFrame
	SchemaServiceLocator	class	extends org.apache.axis.client.Service, implements SchemaService

3.1.4.2 TMS Web Services Server-Side Deployment and Execution

The TMS Web Services are deployed on the server by extracting the files in TMS-WS.zip file. The files should be extracted into a folder, TMS, created in the extraction. This file should be placed in the webapps directory of the server's Tomcat application. Tomcat must then be restarted, after which the Web Services will be deployed at the URL of the server: http://localhost:8080/TMS/services/SchemaService. (The port may differ, depending on the installation settings of Tomcat, but port 8080 is prevalent.) The files deployed for the TMS Web Services consist of:

- apache-tomcat-5.5.12\webapps\TMS\index.html
- apache-tomcat-5.5.12\webapps\TMS\META-INF\MANIFEST.MF
- apache-tomcat-5.5.12\webapps\TMS\WEB-INF\classes\com\itt\tms\ws\*.class (three files)
- apache-tomcat-5.5.12\webapps\TMS\WEB-INF\classes\log4j.properties
- apache-tomcat-5.5.12\webapps\TMS\WEB-INF\classes\SchemaService.wsdl
- apache-tomcat-5.5.12\webapps\TMS\WEB-INF\lib\*.jar (10 files)
- apache-tomcat-5.5.12\webapps\TMS\WEB-INF\server.config.wsdd
- apache-tomcat-5.5.12\webapps\TMS\WEB-INF\web.xml

The server configuration includes the definition of the directory in which TMS Schemas are stored. A context parameter (Figure 3-7) is defined in the web.xml file deployed as described above. To configure the TMS Web Services for your server, change the value of the context parameter to match the location of the Schemas in your installation of the TMS.

```
- <context-param>
        <param-name>XSDPath</param-name>
        <param-value>C:/XSDs/</param-value>
  </context-param>
```

Figure 3-7: Default Definition of Location of TMS Schemas

One can check that the TMS Web Services are available, once they have been deployed and Tomcat has been restarted. Point your Web browser to the following URL: http://localhost:8080/TMS/services/SchemaService?WSDL. If the TMS Web Services have been correctly configured, the server should return the Web Services Definition Language (WSDL) for the TMS Web Services, as shown in Figure 3-8.

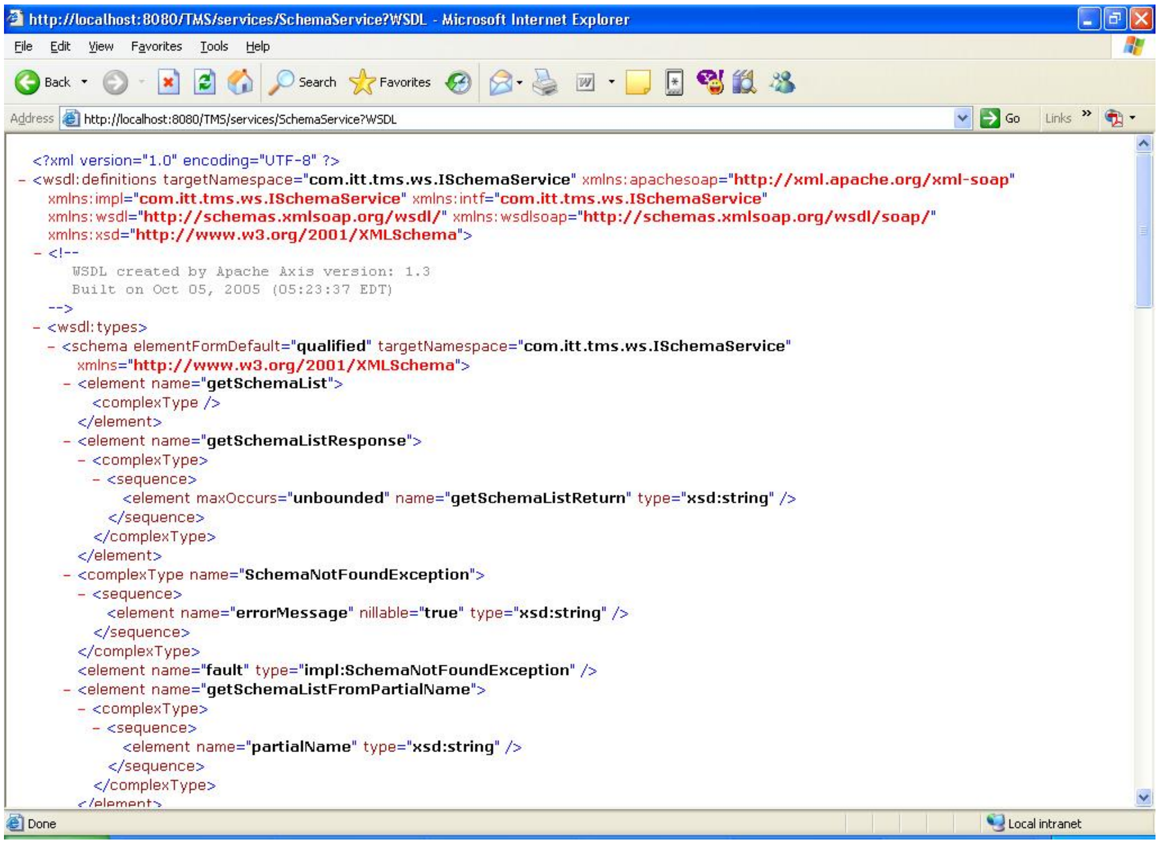
3.1.4.3 Client Deployment

The TMS Web Services software includes a default Web Services client. This default client, excluding the Web Services client GUI, was generated using WSDL-to-Java, an Apache Axis utility. Similar tools for generating Web Services clients are available for C++ and other languages.

To deploy the default client, copy the "TMSClient.jar" file into a newly created folder named, say, TMSClient. This jar file is all that needs to be present for TMS Web Services client.

3.1.4.4 Client Execution

From a MS-DOS window, change the working directory to be TMSClient. Your Java CLASSPATH variable should include the current directory. Execute the command "java –jar TMSClient.jar". A window should open, as shown in Figure 3-9. Each one of the three Web Services can be invoked by pushing the corresponding button at the bottom of the window. The first two Web Services, to search for Schemas and to get a specified Schema, each require a text string as an argument. That argument should be entered in the box at the top of the window.


http://localhost:8080/TMS/services/SchemaService?WSDL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/TMS/services/SchemaService?WSDL
Go Links
Done
Local intranet

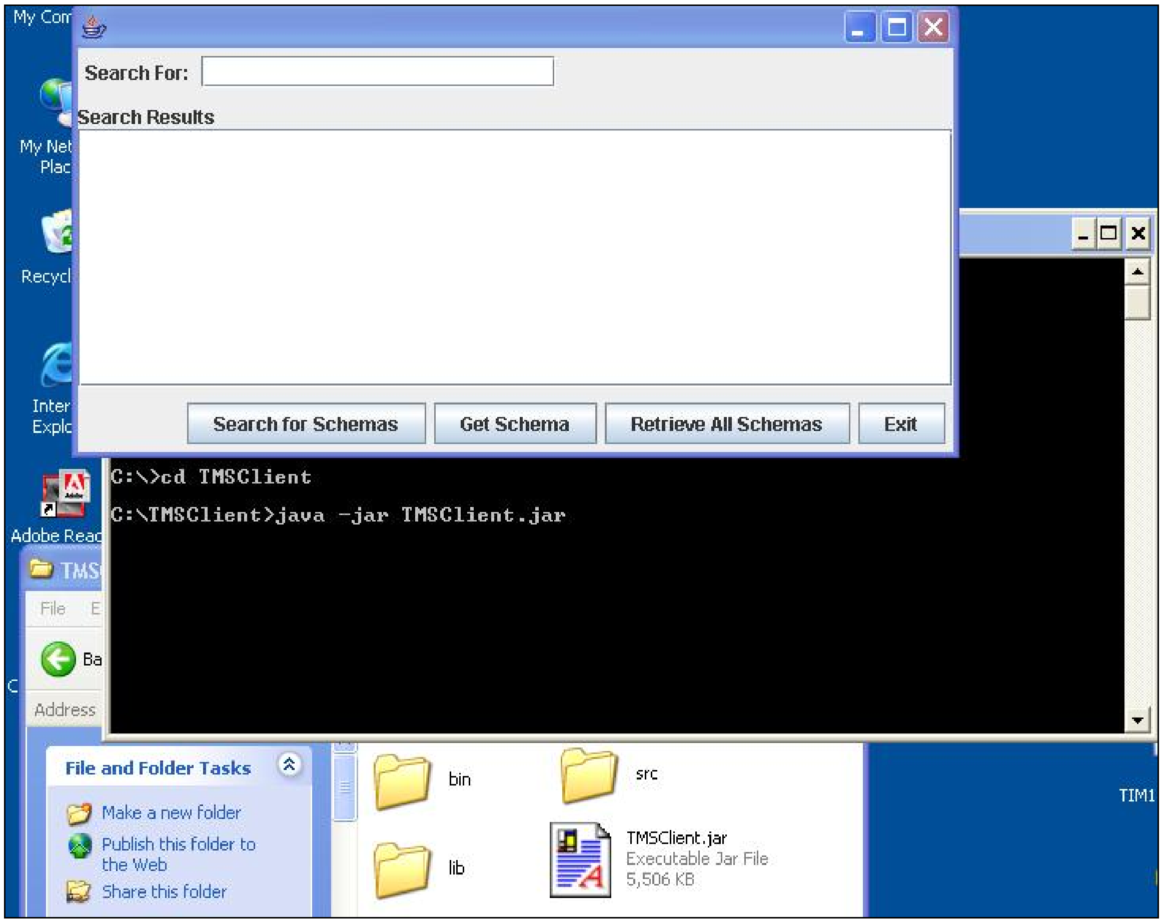
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="com.itt.tms.ws.ISchemaService" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="com.itt.tms.ws.ISchemaService" xmlns:intf="com.itt.tms.ws.ISchemaService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <!--
       WSDL created by Apache Axis version: 1.3
       Built on Oct 05, 2005 (05:23:37 EDT)
    -->
  - <wsdl:types>
    - <schema elementFormDefault="qualified" targetNamespace="com.itt.tms.ws.ISchemaService"
        xmlns="http://www.w3.org/2001/XMLSchema">
      - <element name="getSchemaList">
          <complexType />
        </element>
      - <element name="getSchemaListResponse">
        - <complexType>
          - <sequence>
              <element maxOccurs="unbounded" name="getSchemaListReturn" type="xsd:string" />
            </sequence>
          </complexType>
        </element>
      - <complexType name="SchemaNotFoundException">
        - <sequence>
            <element name="errorMessage" nillable="true" type="xsd:string" />
          </sequence>
        </complexType>
        <element name="fault" type="impl:SchemaNotFoundException" />
      - <element name="getSchemaListFromPartialName">
        - <complexType>
          - <sequence>
              <element name="partialName" type="xsd:string" />
            </sequence>
          </complexType>
        </element>
```

Figure 3-8: The TMS WSDL Returned From Server

3.2 Schema Definition and Maintenance Process

This section describes schema processes supported by schema definition tools. The TMS is a prototype designed to assist in understanding schema definition and maintenance processes and the requirements of tools providing automated support to such processes. The TMS provides benefits to COI information engineers and to their managers following these processes.

The TMS is designed to support configuring information management tools, such as IMCS or NCES, in a COI. The ICED project is premised on the assumption that such tools are configured by means of XSD files, and that this configuration activity is one necessary task in setting up a COI. A COI information engineer is defined, in the context of the ICED project, as a person who is tasked with configuring, or maintaining the configuration of, information management tools.



```
C:\>cd TMSClient

C:\TMSClient>java -jar TMSClient.jar
```

Figure 3-9: The TMS Web Services Client

The TMS supports the COI information engineer, that is, a typical TMS user, by allowing him to concentrate on the domain or application area of concern to the COI. The TMS user is assumed to have:

- Domain knowledge (for example, of tactical military applications)
- Some exposure to introductory computer science.

The TMS user is expected to know that variables have types and that data types can be defined in hierarchies, where fields in one data type are themselves of some defined type. This knowledge is helpful in conceptually understanding what one is doing in interactions with the TMS GUI. On the other hand, the TMS user does not necessarily need to know ontological engineering, XML, or XSD. Advanced users who may know XML and XSD will still find the TMS useful in that it allows them to quickly generate valid XSD files and to visualize their schemas more easily than is possible in examining a raw XSD file.

The manager of a COI information engineer benefits from the TMS in that the TMS allows him to select information engineers from a wider range of candidates. He can now choose information engineers whose area of expertise is more in the COI application areas, instead of in XML and XSD. The TMS allows him to more quickly have such candidates applying their skills to COI issues, without first having gone through training in XSD development. The TMS allows users to build up and share libraries of Elements and Fragments. The contents of these libraries can be reused in different Models, and the TMS GUI allows both information engineers and others to quickly and easily understand the structure of domain data. The manager of information engineers benefits from these capabilities in that the TMS allows the sharing of work products more easily among those reporting to him and with other groups. Given the TMS Web Services, some of this sharing can even be done in machine-to-machine information exchange.

The TMS supports processes for developing schemas. Figure 3-10, based on a figure in a textbook on ontological engineering (Gomez-Perez et al 2004), illustrates schema development processes and how they fit into a larger set of processes. The development of a schema is intended to produce an XSD file, in the case of the TMS, for use in some later process. The TMS Web Services supports feeding the output of schema development processes into use processes. The TMS can support the development of a schema in iterative processes, in which the schema becomes increasingly detailed in later iterations. Figure 3-10 suggests four such iterations. During maintenance processes, the TMS user can use the TMS to correct mistakes discovered in schemas, change schemas to support new capabilities, or to adapt schemas to changes in the environment in which schemas are used.

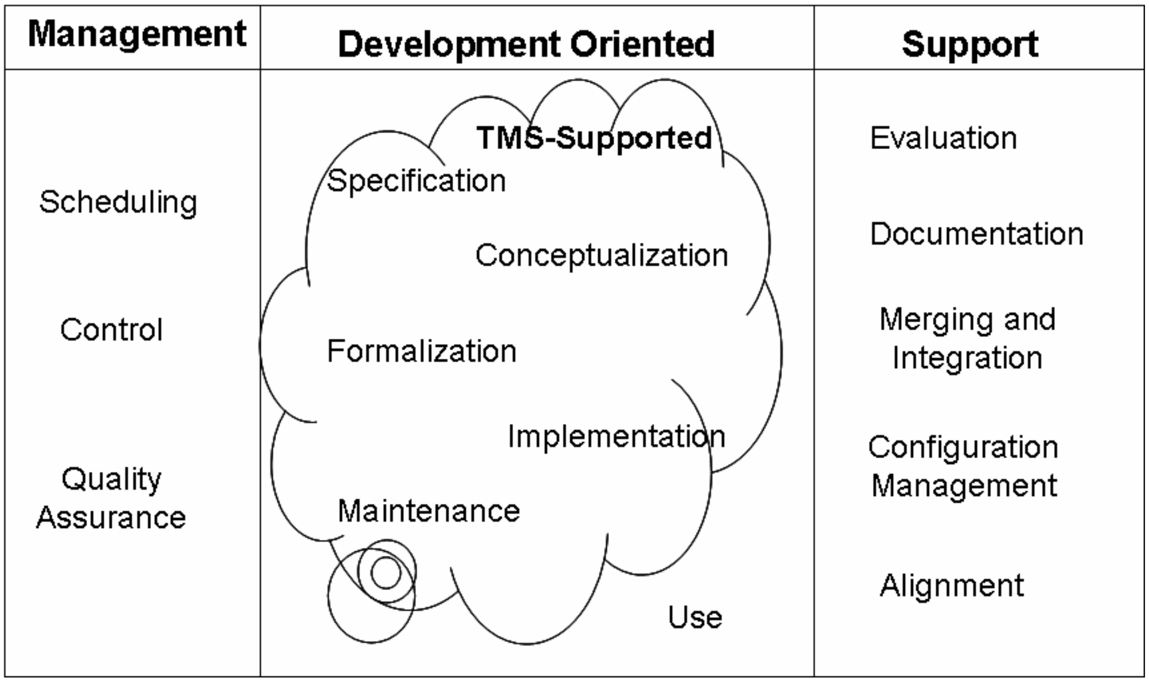


Figure 3-10: Schema Processes

As information management tools become more ubiquitous, the management of schema development will become more important. Management conducts it owns processes. These processes include scheduling and planning schema development processes, controlling schema development processes in ensuring that they are preformed on schedule and within budget, and assuring that schema development processes following defined processes. Schema development processes also require various supporting processes. One supporting process would assess whether a developed schema meets its requirements. Schemas must be documented. Schemas, Models, Fragments, and Elements created with the TMS might exist in different versions, and a configuration management process would help in keeping track of different versions. To support reuse, Schemas, Models, Fragments, and Elements might need to be aligned with standards and style guides required before entry into reusable libraries.

The TMS supports implementing each iterative development process as either a top-down or a bottom-up process (Figure 3-11). The TMS allows one to save Models and Fragments with no children and properties. This capability is needed to support the top-down development process shown. Likewise the TMS allows the user to save Elements that are not children of any Fragments that have yet been defined and Fragments that are not yet children of any defined Models. These are necessary capabilities for a bottom-up development process. In either process, the TMS allows the user to reuse and modify existing Models, Fragments, and Elements. In this way, the TMS supports an exploratory style of development.

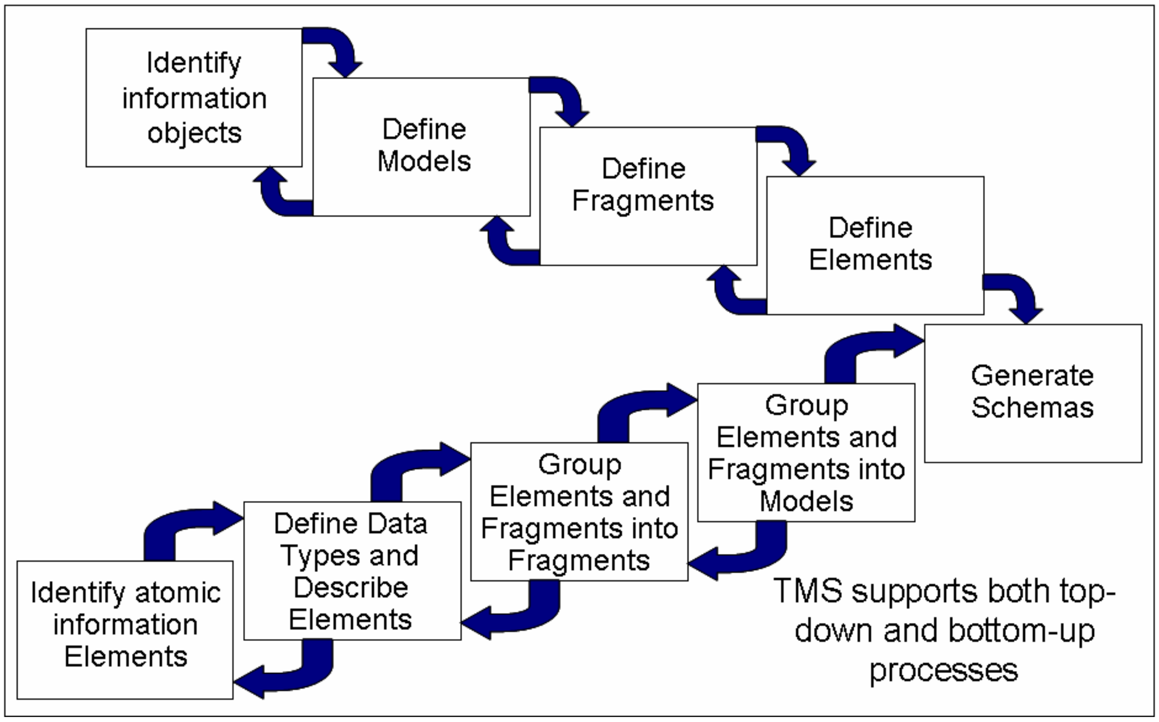

Identify information objects
Define Models
Define Fragments
Define Elements
Generate Schemas
Group Elements and Fragments into Models
Group Elements and Fragments into Fragments
Define Data Types and Describe Elements
Identify atomic information Elements
TMS supports both top-down and bottom-up processes

Figure 3-11: Top-Down and Bottom-Up Schema Development Processes

The TMS is a proof of concept showing that appropriate automated support can ease the cognitive burden in configuring information management tools for a COI. The TMS does not support a full implementation of XSD or a full description of ontologies. Some XSD capabilities not available in the XSDs generated by the current version of the TMS may be useful or required for COI information engineering. The experience of the TMS developers suggests that if the TMS were extended to fully implement XSD standards (e.g. W3C 2004), it would become increasingly difficult to encapsulate complex syntax and semantic aspects of XML and XSD. For example, ITT, fairly late in this effort, added a capability to associate regular expressions with properties defined for each Fragment or Model. Regular expressions are merely strings in the TMS; the user is not assisted appreciably in the understanding of how to construct a regular expression. One is taking a risk if one concludes from this research that the full education requirements of a COI information engineer does not require XML and XSD training, since full tool support is not conclusively provided by the TMS to give the engineer all needed capabilities without the engineer ever examining XML and XSD code.

3.3 Scenario and Demonstration

The ICED demonstration illustrates schema processes, demonstrates selected TMS features, and provides a capability with which any TMS user can demonstrate the TMS to others. The demonstration shows the top-down development of a Model, the generation of a Schema, and the retrieval of that Schema by the TMS Web Services. The Model development in the demonstration reuses an existing Fragment and illustrates the development of new Fragments. The Model development and XSD generation illustrate capabilities of the TMS GUI. The XSD retrieved by the TMS Web Services can be used to configure information management tools, assumed to be available in the demonstration scenario to the TMS user. This demonstration can be run, using the information in this report, without ITT support.

3.3.1 Demonstration Scenario

The scenario defined for the demonstration provides a setting in which the TMS user actions conducted during the demonstration are plausible. The scenario presents the actions of a COI information engineer with domain knowledge of tactical military operations that are planned to be conducted in the COI. The scenario illustrates that the COI information engineer does not need knowledge of XML or XSD to use the TMS.

In the scenario, the COI is temporarily isolated from NCES and the DDMR. Perhaps the internal components of the COI are being set up, while some issue with remote communications is being investigated. The scenario postulates the COI information engineer has a rich collection of information management tools, and these are configured to transmit geographical information, but in non-Cursor On Target (COT) format. The TMS is assumed to contain a Schema, decomposed for this geographical information prior to the start of the demonstration. This Schema (Figure 3-12) has latitude and longitude specified in terms of degrees, minutes, and seconds; the COT schema requires a single real number for each dimension. The *detail* Fragment in this existing Schema can be reused in the COT Schema. The COT Schema also requires the definition of various Properties or XML attributes not in the existing Schema for geographical information.

In the demonstration scenario, the COT information engineer is tasked with creating a COT Schema (Figure 3-13). The COI information engineer is assumed to have access to a rich set of information management tools in the COT. The COT Schema created with the TMS can be used to configure some of these tools. For example, the COT Schema can be used to specify information objects to be transmitted between components within the COI. Or the COT Schema, in combination with the existing Schema might be used to configure a fuselet to convert from information objects described by the COT Schema to information objects described by the pre-existing Schema.

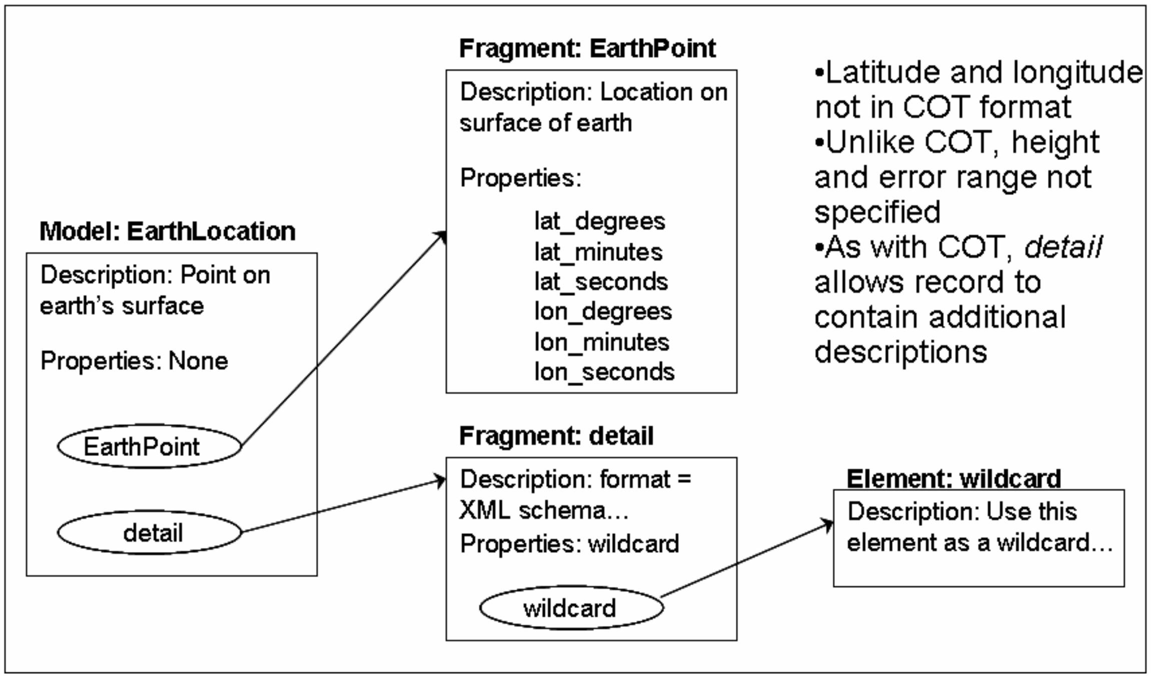

Fragment: EarthPoint
Description: Location on surface of earth
Properties:
lat_degrees
lat_minutes
lat_seconds
lon_degrees
lon_minutes
lon_seconds
•Latitude and longitude not in COT format
•Unlike COT, height and error range not specified
•As with COT, *detail* allows record to contain additional descriptions
Model: EarthLocation
Description: Point on earth's surface
Properties: None
EarthPoint
detail
Fragment: detail
Description: format = XML schema…
Properties: wildcard
wildcard
Element: wildcard
Description: Use this element as a wildcard…

Figure 3-12: Initial Model, Fragments, and Element In Demonstration

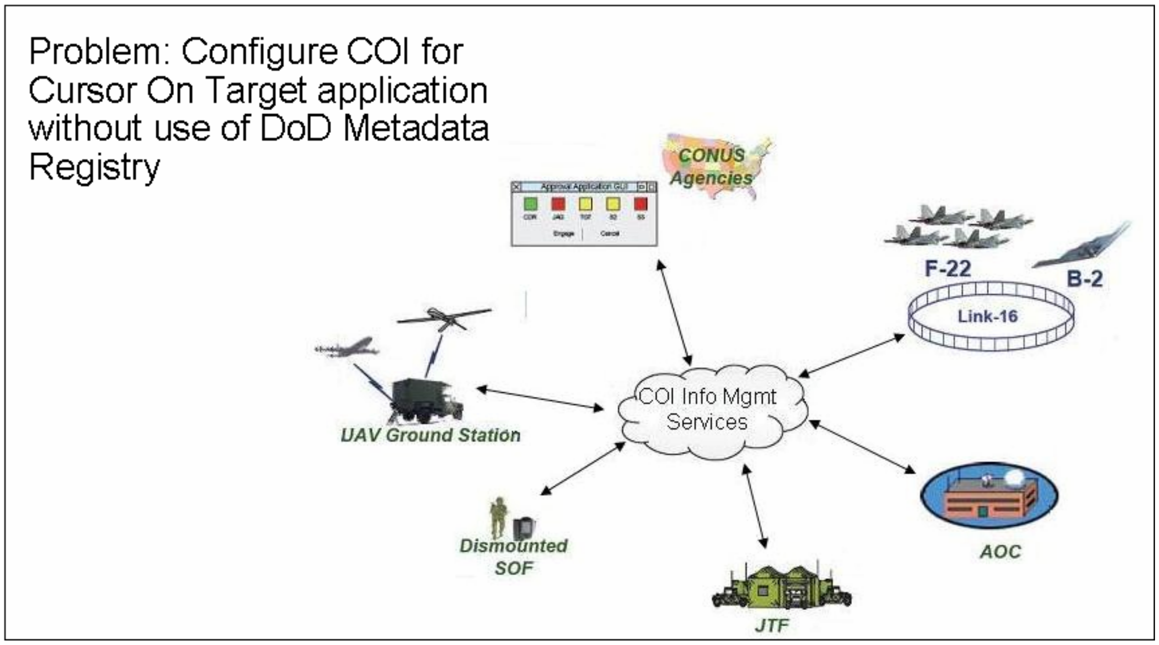

Problem: Configure COI for Cursor On Target application without use of DoD Metadata Registry
CONUS Agencies
F-22
B-2
Link-16
COI Info Mgmt Services
UAV Ground Station
Dismounted SOF
JTF
AOC

Figure 3-13: COI Is Configured In Scenario

3.3.2 Demonstration Deployment

The TMS demonstration data is deployed in the ICEDDemo.zip file. The demonstration data is extracted into a folder DemoData, created in the extraction. The demonstration data is organized into two folders:

- DemoData\DemoStart: The folders and files for configuring the TMS before the demonstration begins.
- DemoData\DemoEnd: The folders and files for configuring the TMS to display the full COT Model during the demonstration.

To configure TMS to display the demonstration data at the appropriate point in the demonstration, delete the files and folder in Eclipse\plugins\iced and copy the files and folders in either DemoStart or DemoEnd to Eclipse\plugins\iced. Further details about when to configure the TMS with the demonstration data is provided in Section 3.3.3.

3.3.3 Demonstration Steps

Before beginning the demonstration, the TMS must be configured to contain the initial data used in the demonstration. To begin the configuration, delete the files and folder in the Eclipse\plugin\iced directory. Then copy the file tms.properties, and the folders, including their contents, from DemoData\DemoStart to Eclipse\plugins\iced, to configure the TMS as described in Section 3.1.3.2. Start up the TMS Web Services server, as described in Section 3.1.4.2. This completes the demonstration configuration.

The ICED demonstration consists of 11 steps:

1. Browse initial TMS configuration
2. Create COT *event* Model
3. Reuse existing *detail* Fragment as *event* child
4. Create COT *point* Fragment
5. Append *point* Fragment to *event* children
6. Edit *event* to ensure children are unordered
7. Define *version* property for *event* Model
8. Define *lat* property for *point* Fragment
9. To skip over defining each Property, restore complete COT *event* Model and browse properties
10. Generate XSD
11. Demonstrate Web Services Schema browsing

3.3.3.1 Browse Initial TMS Configuration

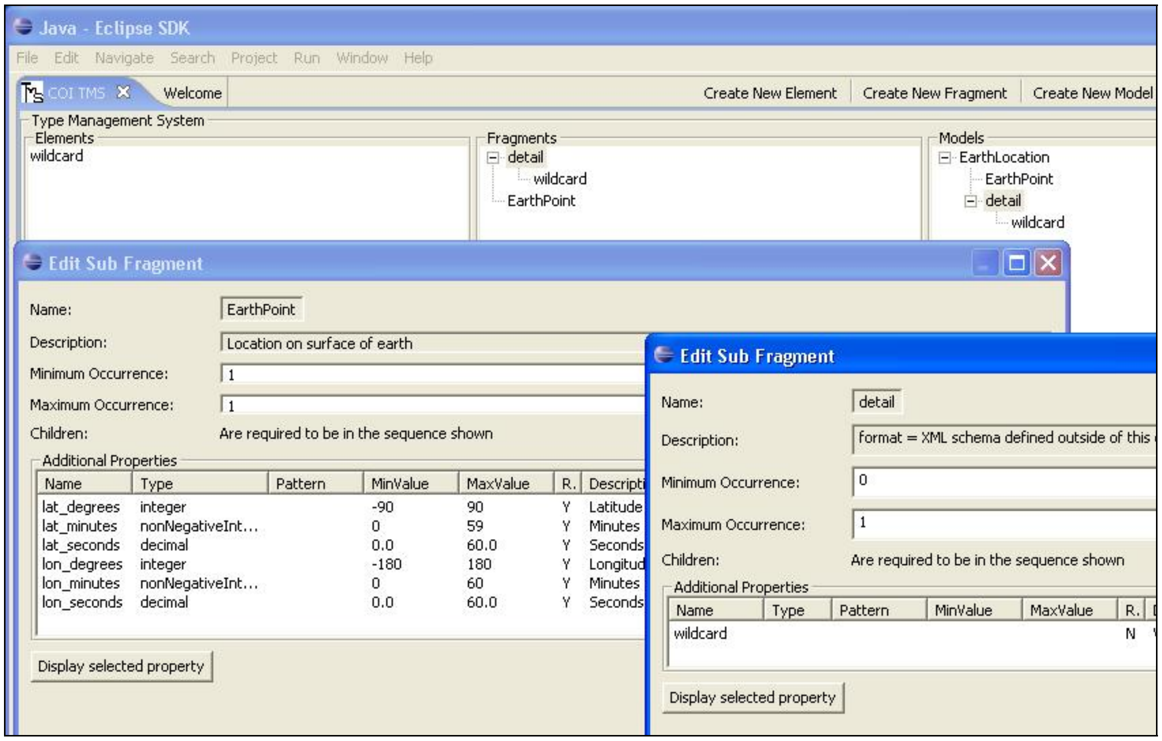
Display the COI TMS configuration, as described in Section 3.1.3.3. You will see that a single Model, *EarthLocation*, is defined. Edit *EarthLocation* by right-clicking on its name and selecting “Edit” from the popup menu that appears. You can point out that

EarthLocation has no properties, and its children are unordered. When you are done examining the EarthLocation window, click on the "Cancel" or "OK" button in the lower right. By clicking on the "plus" next to *EarthLocation*, you can show that it has two children, *EarthPoint* and *detail*. Since these children names appear in the Fragment window, these children are both Fragments.

Edit *EarthPoint* by right-clicking on its name in either or both the Model window and the Fragment window. In the Model window editing, you can see that exactly one *EarthPoint* Fragment must appear in the *EarthLocation* Model. You can see how the ranges of the properties of this Fragment are specified, and how different types are appropriate for different ranges.

Edit the *detail* Fragment. You can see that it has one property, *wildcard*, which cannot be edited. You can also see that the *detail* Fragment has one child, *wildcard*.

At this point, you have examined the Model initializing the demonstration (Figure 3-14). The user in the scenario would like to reuse the *detail* Fragment in the COT *event* Model, needs properties not in the initial Model, and needs to specify the location on the earth in a different format than in the *EarthLocation* Model. The remaining demonstration steps create a Model in which these needs are met.



Name: EarthPoint
Description: Location on surface of earth
Minimum Occurrence: 1
Maximum Occurrence: 1
Children: Are required to be in the sequence shown
Additional Properties

Name	Type	Pattern	MinValue	MaxValue	R.	Descripti
lat_degrees	integer		-90	90	Y	Latitude
lat_minutes	nonNegativeInt...		0	59	Y	Minutes
lat_seconds	decimal		0.0	60.0	Y	Seconds
lon_degrees	integer		-180	180	Y	Longitud
lon_minutes	nonNegativeInt...		0	60	Y	Minutes
lon_seconds	decimal		0.0	60.0	Y	Seconds

Name: detail
Description: format = XML schema defined outside of this
Minimum Occurrence: 0
Maximum Occurrence: 1
Children: Are required to be in the sequence shown
Additional Properties

Name	Type	Pattern	MinValue	MaxValue	R.
wildcard					N

Figure 3-14: Browsing Fragments in the Earth Location Model

3.3.3.2 Create COT *event* Model

Click on the “Create New Model” button on the upper right of the COI TMS view. In the window that appears enter “event” as the name and “Event Definition” as the description. When done, click on the “OK” button on the lower right. The name of your newly-created *event* Model should now be displayed in the Model window.

3.3.3.3 Reuse *detail* Fragment as *event* Child

Right-click on *detail* in the Model window, and select “Add Fragment to Model” from the popup menu that appears. A window will appear in which the two Fragments, *detail* and *EarthPoint* are listed. Click on *detail* in that window, and it will become highlighted. Click “OK”.

A new window for the *detail* child then appears. Enter 0 for the “Minimum Occurrence”, and accept the default value of unity for the “Maximum Occurrence” by clicking on the “OK” button.

3.3.3.4 Create COT *point* Fragment

Click on the “Create New Fragment” button on the upper right of the COI TMS view. In the window that appears enter “point” as the name and “Point” as the description. When done, click on the “OK” button on the lower right. The name of your newly-created *point* Fragment should now be displayed in the Fragment window.

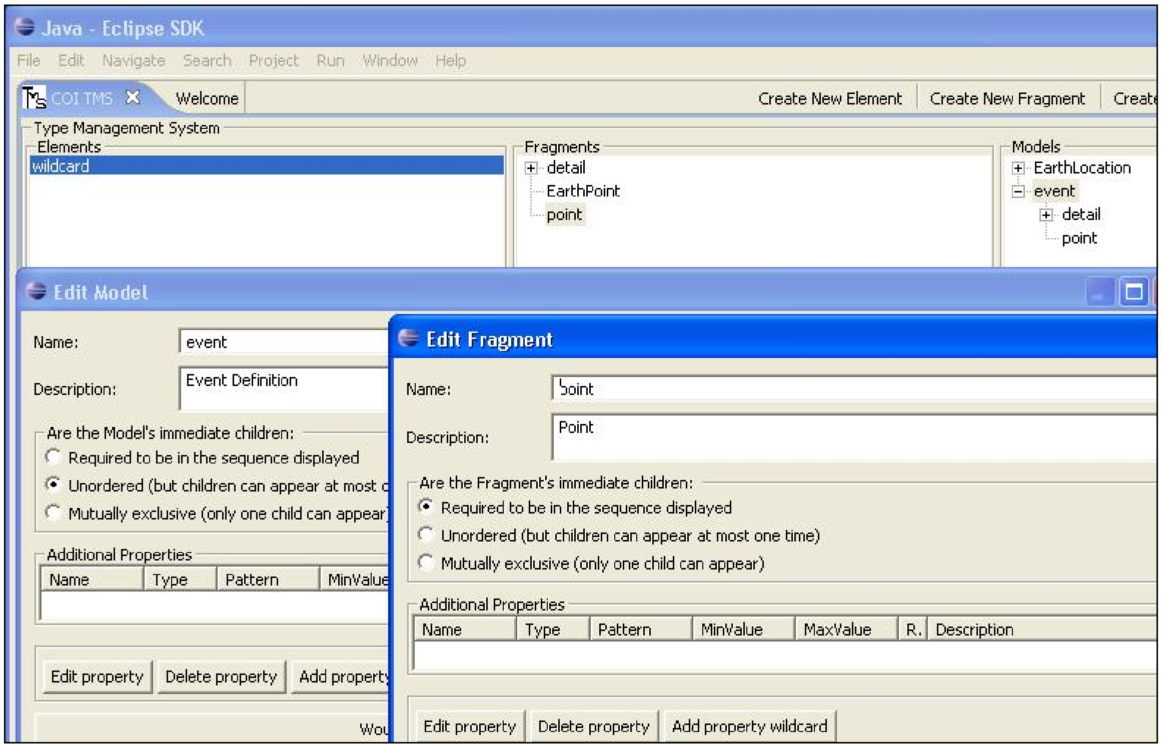
3.3.3.5 Append *point* Fragment to *event* Children

The *point* Fragment can be made a child of the *event* Model in the same way that the *detail* Fragment was, as described in Section 3.3.3.3. For variety, you can demonstrate the drag and drop capability of the TMS GUI. Both the “Minimum Occurrence” and the “Maximum Occurrence” of *point* should be the default value of unity.

3.3.3.6 Edit *event* To Ensure Children Are Unordered

In an instance of the COT *event* Schema, the children Fragments can appear in any order. At most one instance of each child can appear. Displaying the TMS GUI help capabilities is not a formal step in the demonstration. This step, however, might be a good opportunity to bring up the help system, following the guidance in Section 3.1.3.3. What it means for a Model’s children to be unordered is explained in the “Create New Model” explanation under TMS help

To ensure the COT *event* Model’s children are unordered, edit the *event* Model by right-clicking on its name in the Model window, and select “Edit” from the popup menu that appears. Select “Unordered” in the radio button menu for the Model’s children in the window that appears. Click the “OK” button when finished. Figure 3-15 shows a display when browsing the TMS data at this point in the demonstration.


Java - Eclipse SDK

File Edit Navigate Search Project Run Window Help

COI TMS | Welcome | Create New Element | Create New Fragment | Create

Type Management System

Elements: wildcard

Fragments: detail, EarthPoint, point

Models: EarthLocation, event (detail, point)

Edit Model

Name: event

Description: Event Definition

Are the Model's immediate children:
- Required to be in the sequence displayed
- [✓ Unordered (but children can appear at most o]
- Mutually exclusive (only one child can appear)

Additional Properties

Name	Type	Pattern	MinValue

Edit property | Delete property | Add property

Edit Fragment

Name: point

Description: Point

Are the Fragment's immediate children:
- [✓ Required to be in the sequence displayed]
- Unordered (but children can appear at most one time)
- Mutually exclusive (only one child can appear)

Additional Properties

Name	Type	Pattern	MinValue	MaxValue	R.	Description

Edit property | Delete property | Add property wildcard

Figure 3-15: *event* Model has *point* and *detail* Fragments As Unordered Children

3.3.3.7 Define *version* Property For *event* Model

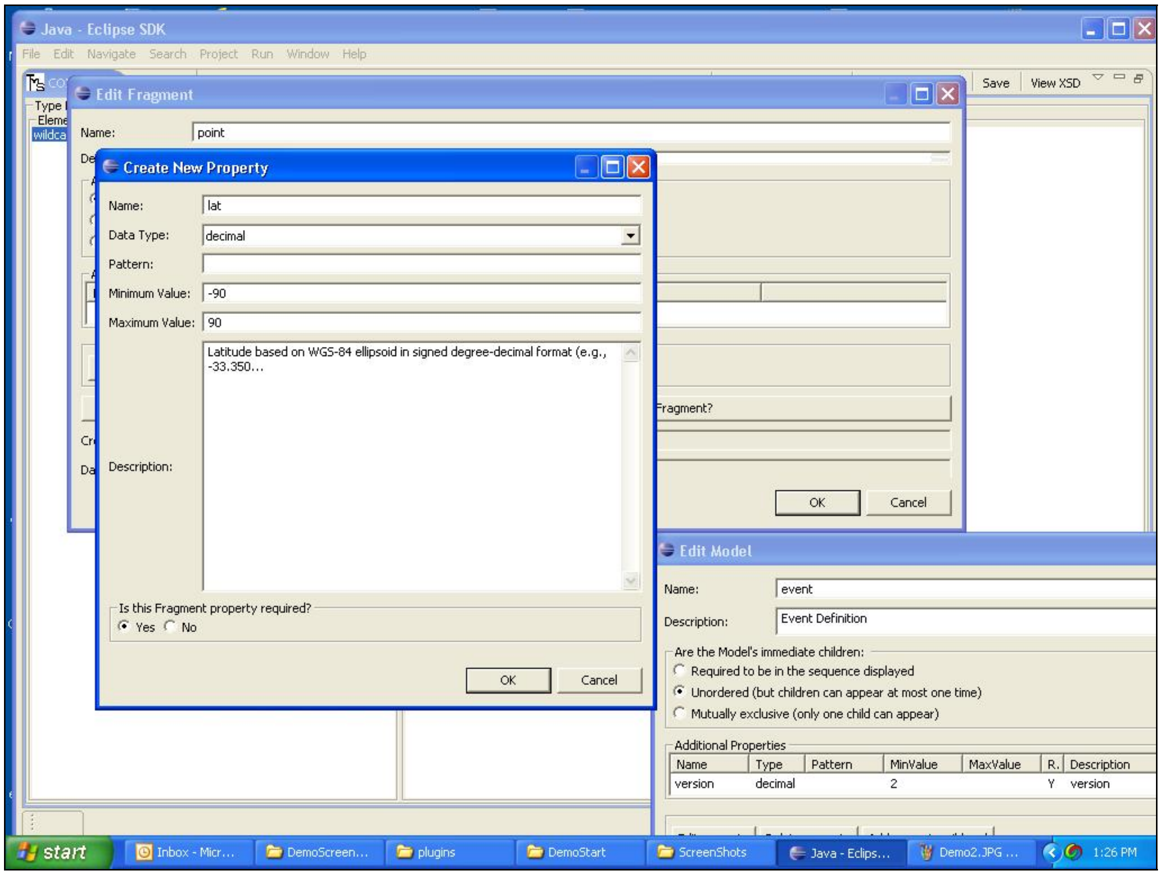
Once again, edit the *event* Model. Click on the button labeled “Would you like to enter additional properties pertaining to this Model?”

Enter “version” for name. Choose the decimal type for the Data Type. Leave pattern and minimum value blank. Enter a maximum value of 2, enter a description of “version”, and specify that this property is required. When done, click “OK”.

3.3.3.8 Define *lat* Property For point Fragment

Edit the *point* Fragment from the Fragment window. Click on the button labeled “Would you like to enter additional properties pertaining to this Fragment?”

Enter “lat” for name. Choose the decimal type for the Data Type. Leave pattern blank, enter a minimum value of -90, enter a maximum value of 90, enter a description of “Latitude based on WGS-84 ellipsoid…”, and specify that this property is required (Figure 3-16). When done, click “OK”.



Edit Fragment — Name: point

Create New Property

Name: lat
Data Type: decimal
Pattern:
Minimum Value: -90
Maximum Value: 90
Description: Latitude based on WGS-84 ellipsoid in signed degree-decimal format (e.g., -33.350...

Is this Fragment property required? [✓ Yes] No

Edit Model

Name: event
Description: Event Definition

Are the Model's immediate children:
- Required to be in the sequence displayed
- [✓ Unordered (but children can appear at most one time)]
- Mutually exclusive (only one child can appear)

Additional Properties

Name	Type	Pattern	MinValue	MaxValue	R.	Description
version	decimal		2		Y	version

Figure 3-16: Defining a Property for *point* Fragment

3.3.3.9 Restore Complete COT *event* Model And Browse Properties

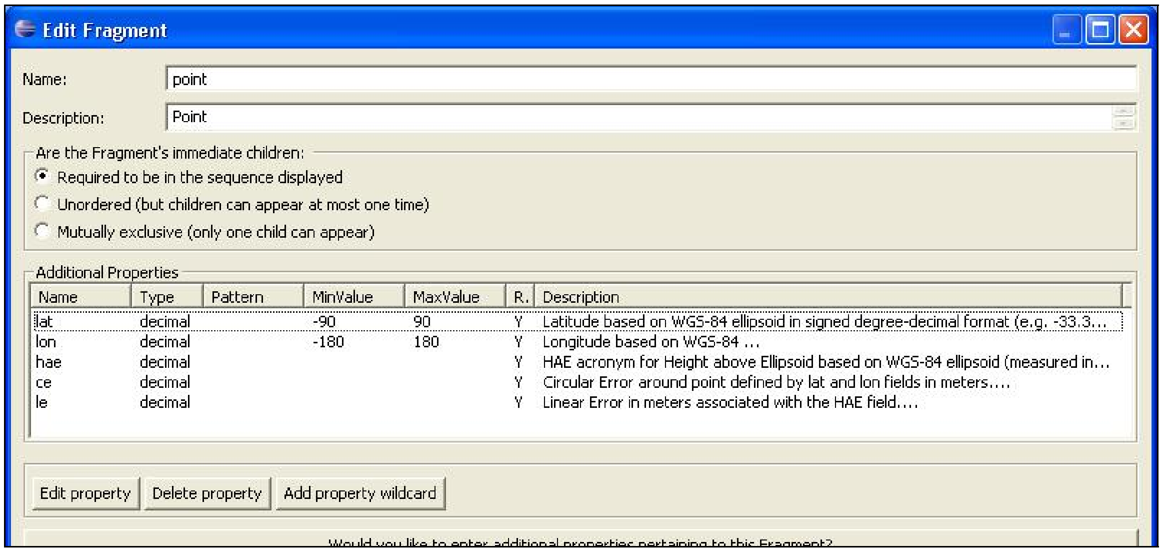
Exit Eclipse, and delete the files and folder in the Eclipse\plugin\iced directory. Then copy the file tms.properties, and the folders, including their contents, from DemoData\DemoEnd to Eclipse\plugins\iced, to configure the TMS as described in Section 3.1.3.2.

Restart Eclipse and display the COI TMS view. You should browse the COT *event* Model, in the same way you browsed the *EarthLocation* Model in the first step of the demonstration (Section 3.3.3.1). Note that both the *event* Model and the *point* Fragment have several more properties defined. (Figure 3-17 shows the properties for the *point* Fragment.) This step of restoring the TMS database is merely a shortcut to defining all of these properties.

3.3.3.10 Generate XSD

Right-click on the name of the *event* Model in the Model window, and select “Generate and Save XSD” from the popup menu that appears. You can browse the XSD file created by the TMS by clicking on “View XSD” on the upper right of the COI TMS view. The

TMS GUI allows a long XSD file to be easily browsed and displayed in a compact form to the user.



Name: point

Description: Point

Are the Fragment's immediate children:
- (●) Required to be in the sequence displayed
- ( ) Unordered (but children can appear at most one time)
- ( ) Mutually exclusive (only one child can appear)

Additional Properties

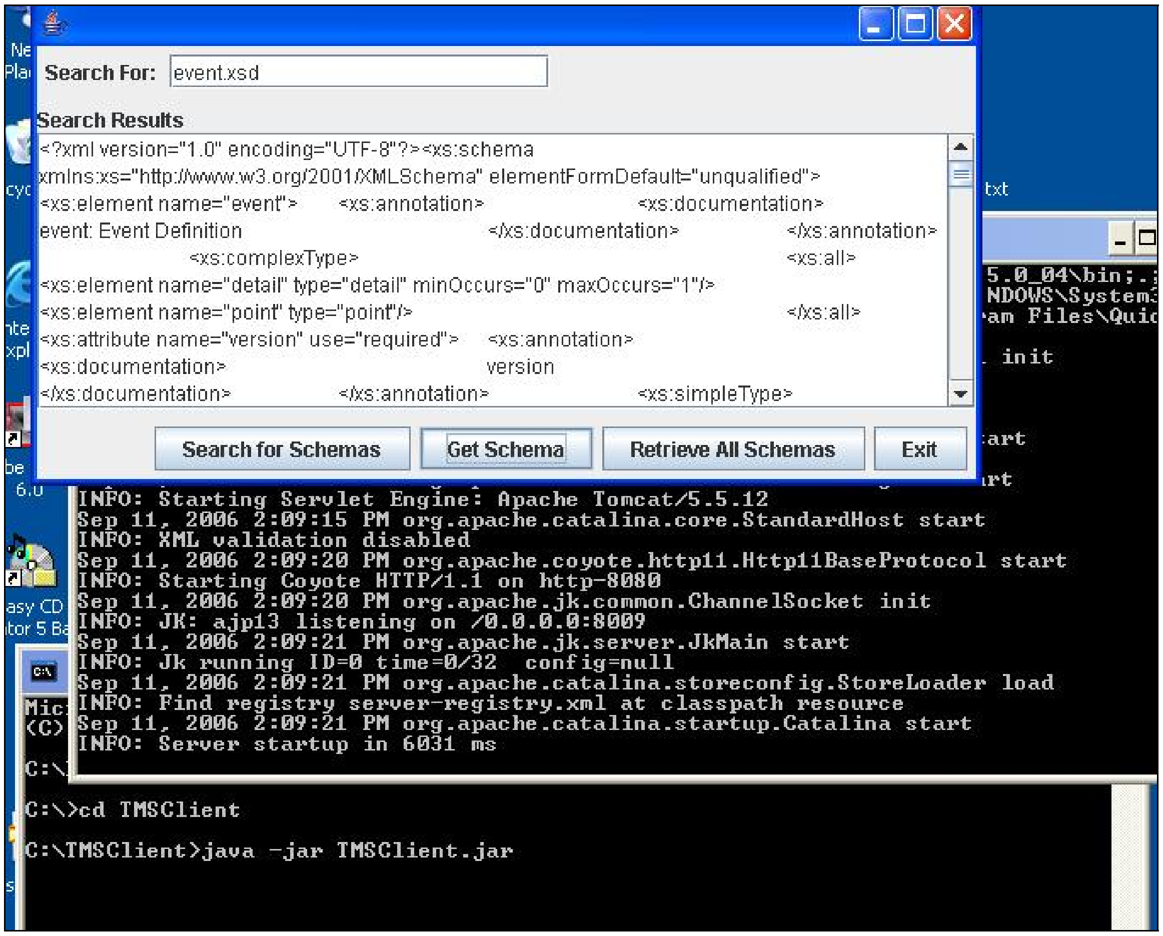
Name	Type	Pattern	MinValue	MaxValue	R.	Description
lat	decimal		-90	90	Y	Latitude based on WGS-84 ellipsoid in signed degree-decimal format (e.g. -33.3...
lon	decimal		-180	180	Y	Longitude based on WGS-84 ...
hae	decimal				Y	HAE acronym for Height above Ellipsoid based on WGS-84 ellipsoid (measured in...
ce	decimal				Y	Circular Error around point defined by lat and lon fields in meters....
le	decimal				Y	Linear Error in meters associated with the HAE field....

Would you like to enter additional properties pertaining to this Fragment?

Figure 3-17: Properties For *point* Fragment

3.3.3.11 Demonstrate Web Services Schema Browsing

Invoke the TMS Web Services client, as described in Section 3.1.4.4. You should, at least, display the *event* XSD created from the TMS GUI (Figure 3-18). The TMS Web Services provides a capability for other systems to automatically extract XSD files from the TMS. As such, human readability of the XSD returned by the TMS Web Services was not emphasized during TMS development.



```
event.xsd
```

```
<?xml version="1.0" encoding="UTF-8"?><xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
<xs:element name="event">       <xs:annotation>                    <xs:documentation>
event: Event Definition                               </xs:documentation>               </xs:annotation>
                    <xs:complexType>                                                   <xs:all>
<xs:element name="detail" type="detail" minOccurs="0" maxOccurs="1"/>
<xs:element name="point" type="point"/>                                                 </xs:all>
<xs:attribute name="version" use="required">     <xs:annotation>
<xs:documentation>                                    version
</xs:documentation>              </xs:annotation>                    <xs:simpleType>
```

```
INFO: Starting Servlet Engine: Apache Tomcat/5.5.12
Sep 11, 2006 2:09:15 PM org.apache.catalina.core.StandardHost start
INFO: XML validation disabled
Sep 11, 2006 2:09:20 PM org.apache.coyote.http11.Http11BaseProtocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Sep 11, 2006 2:09:20 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Sep 11, 2006 2:09:21 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/32  config=null
Sep 11, 2006 2:09:21 PM org.apache.catalina.storeconfig.StoreLoader load
INFO: Find registry server-registry.xml at classpath resource
Sep 11, 2006 2:09:21 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 6031 ms
```

```
C:\>cd TMSClient

C:\TMSClient>java -jar TMSClient.jar
```

Figure 3-18: XSD for COT Returned By TMS Web Services Client

4.0 Conclusions

Under the ICED project, ITT developed a proof-of-concept software tool, the Type Management System (TMS), to assist in the definition of a COI vocabulary and the construction and management of schemas. A goal of the project was to demonstrate the possibility of a functional tool for COI information engineers that would not require strong XML and XSD knowledge. While this goal can be attained if the tool functionality is limited, ITT found that as capabilities were added to the tool, the user would become increasingly likely to require specific XML, XSD, and computer science knowledge. The XSD specification is such that abstracting all computer science complexities from the user seems to be impossible. In addition to achieving certain research results, ITT identified some directions for future research during the course of this development.

4.1 Achievements

The achievements of the ICED project are easily summarized:

- Produced a prototype TMS to support a COI information engineer in defining XSDs for use in COI information management
- Defined an iterative and hierarchical schema development process supported by the TMS
- Defined a scenario and produced a demonstration of the TMS operating under that scenario
- Identified directions for future research for the development of the TMS and the schema processes supported by the TMS

This research promises to benefit COI information engineers and their managers. After the technology described in this report is matured and transitioned, managers will not require engineers trained in XML and XSD. Information engineers can concentrate on issues in their application areas and data domains, rather than XML and XSD syntax. Managers and engineers can more easily visualize and understand their domain data. Engineers can quickly create and modify descriptions of their domain data using easy to understand schema modeling concepts. Finally, the TMS Web Services illustrates capabilities for machine-to-machine access to schema modeling information, with no requirement for human examination of XSDs.

4.2 Directions for Future Research

The TMS and its demonstration show that COI information engineers configuring information management tools can benefit from automated software tool support. The prototype support provided by the TMS can be improved with research in five areas:

- Expansion of TMS support to additional features of the XSD language
- Additional capabilities of the TMS GUI

- Additional capabilities of the TMS Web Services
- Additional capabilities of the TMS
- Expansion of TMS usefulness by global improvements, including in the processes supported by the TMS and in the environment which it operates.

4.2.1 Expansion of the Subset of XSD Features Implemented in the TMS

Currently, the TMS supports the generation of only a subset of the XSD language. A research question is whether software tool support would still allow a COI information engineer to concentrate on application issues, without worrying about XML and XSD details, if more features of the XSD language were implemented in the TMS. The implementation of the complete set of XSD data types in the TMS is one additional XSD language capability that could be added. The TMS currently provides some simple type constraints on properties for Fragments and Models; future research could add a capability to specify simple type constraints for Elements. In any case, simple type constraints could be expanded to include all options available for XDSs, including global simple type constraints. The TMS does not currently support all XSD elements and attributes, and additional research might explore expanding support here too.

4.2.2 Additional TMS GUI Capabilities

Additional capabilities can be added to the TMS GUI. Undo functionality would be a powerful addition. One might add drag and drop capability to allow the user to order the display of Elements, Fragments, and Models at all levels.

4.2.3 Additional TMS Web Services Capabilities

TMS Web Services currently allow to Web Services to search and list the names of XSD files and to retrieve the contents of a named XSD file. Additional Web Services can expand the capabilities of machine-to-machine interactions with the TMS. Web Services could be added to create Elements, Fragments, and Models. One might also add Web Services to edit existing Elements, Fragments, and Models. Furthermore, network security can be added by certificate and client authentication, as in, for example, NCES Security Services. Along with such network security should come capabilities distinguishing what TMS data different clients have access to.

4.2.4 Additional TMS Capabilities

The TMS data storage capabilities were not emphasized during the ICED project. Future research could add a number of capabilities here. For example, the TMS could be modified to store data, such as Elements, Fragments, and Models, in a relational database. Access to TMS data can be controlled with future ownership and permission functionality. Being able to manage multiple versions of Elements, Fragments, Models, and Schemas would be another capability.

Access to Schemas, Models, Fragments, and Elements created and maintained by the TMS could be provided to other information management tools, outside of a Web Services context. Future research could investigate the creation of a TMS Application Program Interface (API) to provide capabilities for integrating the TMS with other software tools and systems.

Capabilities to load TMS data from other applications imply a need for further functionality. In particular, validity checks should be researched to ensure that TMS data is stored and loaded in a consistent state. Furthermore, a capability to load XSDs not generated from the TMS into the TMS could be useful, especially if Models, Fragments, and Elements can automatically be generated from such loaded XSDs.

4.2.5 Expansion of Usefulness

The TMS currently supports the development of XSD but does not capture ontology information, as coded, for example, in OWL. Future research could explore capabilities to support ontological development and maintenance processes.

The TMS is intended to operate in an environment with rich information management tools, but the ICED project did not investigate interactions with tools and services, either existing or under development. Future research might investigate, for example, the use of the TMS in autonomously registering XSDs with the DDMR and the benefits gained by registering TMS services with the NCES Service Discovery Services.

5.0 References

Boehm, B. W. (1988). “A Spiral Model of Software Development and Maintenance”, *IEEE Computer*, (May): 61-72.

Burnette, Ed (2005). *Eclipse IDE: Pocket Guide*, O’Reilly.

Gallardo, D., E. Burnette, and R. McGovern (2003). *Eclipse in Action: A Guide for Java Developers*, Greenwich: Manning Publications.

Gomez-Perez, A., M. Fernandez-Lopez, and O. Corcho (2004). *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*, London: Springer-Verlag.

Harlow, F. N. (2005). *A JBI Information Object Engineering Environment Utilizing Metadata Fragments for Refining Searches on Semantically-Related Object Types* (Master,s Thesis) AFIT/GCE/ENG/05-03, Air Force Institute of Technology.

ISO/IEC (2004). *International Standard: Information Technology – Metadata Registries,* Second Edition ISO/IEC 11179-1 (15 September)

Lacy, L. W. (2005). *OWL: Representing Information Using the Web Ontology Language*, Trafford.

McGuinness, D. L. and F. van Harmelen (editors) (2004). *OWL Web Ontology Language Overview*, W3C Recommendation, 10 February, http://www.w3.org/TR/owl-features/

Quine, W. V. (1953). “On What There Is”, in *From A Logical Point of View*, Cambridge: Harvard University Press.

W3C (2004). *XML Schema Part 1: Structures Second Edition*, W3C Recommendation 28 October, http://www.w3.org/TR/xmlschema-1/

Appendix A. Acronyms

AFRL/IF	Air Force Research Laboratory, Information Directorate
API	Application Programming Interface
COI	Community Of Interest
COT	Cursor On Target
DAG	Directed Acyclic Graph
DDMR	DoD Metadata Registry
DoD	Department of Defense
DOM	Document Object Model
GUI	Graphical User Interface
ICED	Infosphere Concept Exploration and Development
IDE	Integrated Development Environment
IMCS	Information Management Core Services
MSR	Metadata Schema Repository
NCES	Net-Centric Enterprise Services
OIM	Operational Information Management
OWL	Web Ontology Language
PI	Principal Investigator
RI	Reference Implementation
SAX	Simple API for XML
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SOW	Statement of Work
SWT	Standard Widget Toolkit
TCP/IP	Transmission Control Protocol/Internet Protocol
TIM	Technical Interchange Meeting
TMS	Type Management System
URL	Uniform Resource Locator
WS	Web Services
WSDL	Web Services Definition Language
XML	eXtensible Markup Language
XMLDB	XML Database
XSD	XML Schema Definition

Appendix B: Analysis Of TMS Extensions To Support OWL

How can the TMS be extended to support the capture of ontology information and the generation of OWL schemas? There is not a one-to-one relationship between OWL schema constructs and XSD constructs. Hence, extending the TMS tool to generate OWL schemas is not simple. Although TMS is extensible, the challenge will be to add the ability to capture desired ontology information without making the tool unduly complicated. Adding ontological functionality to the TMS is not a trivial exercise, but nothing in the TMS design precludes addition.

Jena, an open-source Java framework for building applications under the semantic web, might be useful in extending the TMS to support OWL. Jena can be used with Eclipse. Perhaps a “Generate OWL” menu option could be added to the TMS, next to the currently existing “Generate XSD” menu option. The “Generate OWL” menu option would invoke the Jena models.

An example illustrates some of these points. Figure B-1 presents a sample of Models, Fragments, and Elements that can be defined in the TMS. Figure B-2 presents a possible ontological characterization of this example. That is, Fragments and Elements are explicitly shown as classes in the ontology, along with their properties and relations among instances of each. The possibility of this sort of representation justifies the claim that the TMS design does not preclude adding OWL functionality. Figure B-2 constitutes an outline of Jena code for creating a model from which an OWL schema can be generated. ITT also examined Protégé, an open-source ontology editor, for ideas on generating OWL schemas. It too can be used with Eclipse. It was installed and used to define simple Elements and Fragments from which an OWL schema was generated.

An extension of the TMS to support OWL would also need to reconsider how Fragments and Elements are stored. These are currently stored as XML in text files. The structure of this XML would have to be extended to provide for storage of additional ontological information.

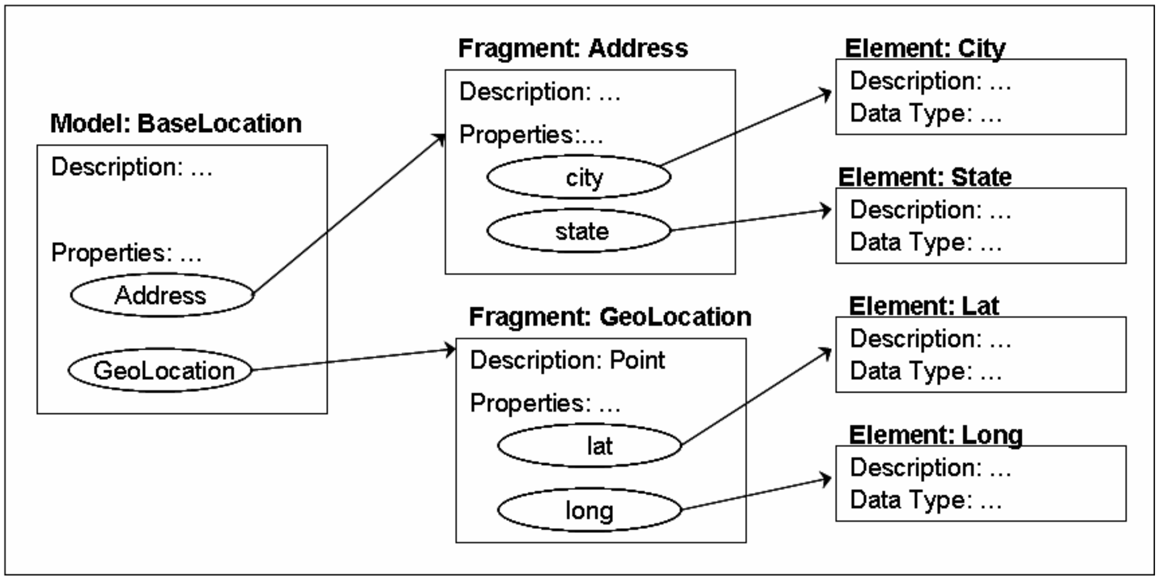


Figure B-1: An Example Model For The TMS

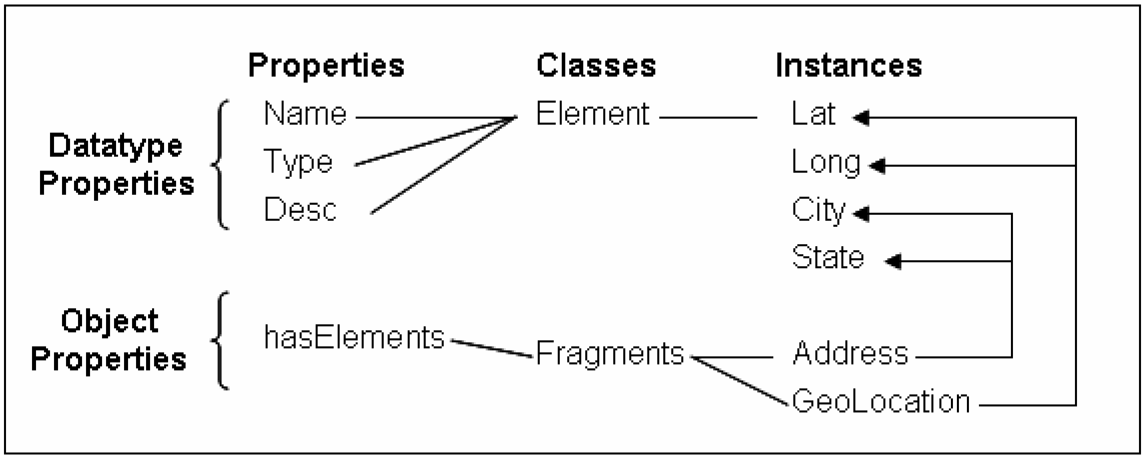


Figure B-2: A TMS Model Interpreted As An Ontology

```
OntModel model = ModelFactory.createOntologyModel();
OntClass element = model.createClass();
OntClass fragment = model.createClass();
DatatypeProperty name = model.createDatatypeProperty(
   arguments );
DatatypeProperty type = model.createDatatypeProperty(
   arguments );
DatatypeProperty desc = model.createDatatypeProperty(
   arguments );
Individual lat = model.createIndividual( element );
Individual long = model.createIndividual( element );
Individual city = model.createIndividual( element );
Individual state = model.createIndividual( element );
ObjectProperty hasElements =
   model.createObjectProperty( arguments );
ObjectProperty hasFragments =
   model.createObjectProperty( arguments );
    // hasFragments is not used in this example
Individual address = model.createIndividual( fragment
   );
Individual geoLocation = model.createIndividual(
   fragment );
// Ontology classes element and fragment are part
//   of model.
// Properties name, type, and desc, and their values
//   should be added to each individual element
//   (lat, long, city, state) created above.
// Property hasElements and its values (city, state)
//   should be added to address
// Property hasElements and its values (lat, long)
//   should be added to geoLocation
```

Figure B-3: Jena 2.4 Code Framework Creating An Example OWL Model

Appendix C: ICED Software Catalog

Component	File	Description
TMS GUI and Storage	iced_1.0.0_src.zip	Source and development files
	iced_1.0.0.zip	Jar file and TMS data files to deploy as Eclipse plugin
TMS Web Services	TMS-WS.zip	Web Services to deploy under Tomcat
	Type Management System.zip	Source and development files for TMS Web Services server and client
	TMSClient.jar	Default Web Services client to deploy
Demonstration	ICEDDemo.zip	TMS data for demonstration
ICED Final Report	ICEDFinalReprt.pdf	This document