CHAPTER III
METHODOLOGY

3.1 Introduction – Prelude

A digital library is a complex information system. The high demand for building digital libraries by non-experts requires a simplified modeling process and rapid generation of digital libraries. To enable rapid generation, digital libraries should be modeled with descriptive languages. A visual modeling tool would be helpful to non-experts so they may model a digital library without knowing the theoretical foundations and the syntactical details of the descriptive language. The CADDTIE method is designed and modeled to enhance the rapid generation in digital libraries.

The CADDTIE method is the generic process traditionally used by instructional designers and digital library content developers. The seven phases- Create; Analysis, Design, Development, Testing, Implementation and Evaluation represent a dynamic, flexible guideline for building effective digital library content tools. Most of the current instructional design methods are spin-offs or variations of the CADDTIE methodology; other methods include the Dick & Carey and the Kemp ISD models. One commonly accepted improvement to this model is the use of rapid prototyping.

3.2 Brief Description of the DL 5S Caddtie Model

DL 5S CADDTIE is a modeling tool that supports domain-specific modeling (digital library modeling). In domain-specific modeling, the model is made up of elements that are part of the domain world, not the whole entity world. DL 5S CADDTIE is tailored to accommodate a certain domain metamodel, the DL 5S model. The methods that are appropriate only to this domain can be used to optimize the modeling process. Reuse in a specific domain is also more realistic and efficient, because the models in one domain have more characteristics in common.
The information gain ratio is just the ratio between the information gain and the intrinsic value. Figure 3.1 shows the DL5S CADDTIE analysis model. Table 3.1 gives the data about the DL5S CADDTIE primitives and its implementation. These ideas of receiving continual or formative feedback while instructional materials are being created. This method attempts to save time. For example, the CADDTIE model can be used in the framework for helping to create new research topics in learning technology in digital library users.

![Diagram](image)

Figure 3.1 DL 5S CADDTIE analysis

Table 3.1 DL 5S CADDTIE Primitives and Implementations

<table>
<thead>
<tr>
<th>Model</th>
<th>Primitives</th>
<th>DL 5S CADDTIE Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream model</td>
<td>Text: Video: audio: picture: programme</td>
<td>MIME types</td>
</tr>
<tr>
<td>Structures Model</td>
<td>Collection: catalog: hypertext:document: metadata: organization tools</td>
<td>XML and RDF Schemas; Topics maps ML(XTM)</td>
</tr>
<tr>
<td>Spaces Model</td>
<td>User interface: index: retrieval model</td>
<td>MathML, UIML, XSL</td>
</tr>
<tr>
<td>Scenarios Model</td>
<td>Service :event: condition: action</td>
<td>Extended UML sequence diagrams: XML serialization</td>
</tr>
<tr>
<td>Societies Model</td>
<td>Community : Service managers: actors relationships: attributes: operations</td>
<td>XML serialization</td>
</tr>
</tbody>
</table>
The DL 5S model extensively uses existing standard description languages, for example, UIML. The reason is that the DL 5S model involves many sub-domains, and there have been already many standard specifications for each sub-domain. It is important for the DL 5S language to be able to interoperate with other standards. DL 5S is designed to be able to combine and merge with other standard description languages.

Table 3.2 shows the objectives within a DL 5S CADDTIE

<table>
<thead>
<tr>
<th>Model</th>
<th>Formal Definition</th>
<th>Objectives within DL 5S CADDTIEModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream model</td>
<td>Sequences of arbitrary</td>
<td>Describe properties of the DL content such as encoding and language for textual material or particular forms of multimedia data</td>
</tr>
<tr>
<td>Structures Model</td>
<td>Labeled directed graphs</td>
<td>Specify organizational aspects of the DL (e.g., structural/descriptive metadata, hypertexts, taxonomies, classification schemes)</td>
</tr>
<tr>
<td>Spaces Model</td>
<td>Sets of objects and operations on those objects that obey specific constraints</td>
<td>Define logical and presentational views of several components</td>
</tr>
<tr>
<td>Scenarios Model</td>
<td>Sequences of events that modify states of a computation in order to accomplish some functional requirements</td>
<td>Detail the behavior of the DL services</td>
</tr>
<tr>
<td>Societies Model</td>
<td>Sets of communities and relationships among them</td>
<td>Define managers, responsible for running DL services: actors, that use those services: and relationships</td>
</tr>
</tbody>
</table>

There are also many well-developed tools for those sub-domains. For example, metadata is an important element in the DL 5S model. Many important metadata standards have been widely accepted, such as ETD-MS Atkins (2001)\(^1\),
Dublin Core, IMS and MARC. Several existing metadata editors can be used to view and edit metadata. Another example is in the scenario part of the DL 5S model. A specific scenario can be modeled and described by UML sequence diagrams. Existing UML modeling tools can be used for this purpose.

3.3 Problem Statement

Digital library is an integration of many application fields of computer science such as information retrieval, databases, and hypertext. To build a digital library, the specification of the content to be stored, content organized, structured, described, accessed, kinds of services offered (e.g., searching, browsing, personalizing, collaborating); patrons services and interaction with the Digital library environment Gonçalves (2002)². Until now, none of those questions have been answered perfectly in previous researches. Accordingly, it is difficult and time consuming to build a new system right now.

Libraries are undergoing tremendous changes. On one side, they are facing three major challenges – shrinking budget, shortage of space and increasing cost of publications, and on the other side, we face the challenges posed by advances in the field of ICT. The remarkable growth of electronic information in the last few decades has changed the scenario and solved the problem of space. In this electronic era, digitized information is available on CDs, audio and video cassettes etc., as well as on the internet. E-contents play a vital role in the field of science and engineering studies. Access to journals has become an important and valuable tool for researchers, students and faculty.

The user community is becoming more familiar with these tools and now they have started using them very regularly. In India, the Ministry of Human Resource Development (MHRD), has set-up the Indian National Digital Library in Engineering Sciences, and Technology (INDEST) consortium. The INDEST consortium has commenced its operation through its headquarters at the IIT, Delhi since December 2002. Access to E-contents resources is now considered more important than collection building, especially if the access is perpetual in nature.
The access to E-contents for the member institutions under the INDEST consortium has increased to a greater extent. These resources and its impact can be examined elaborately. Technical institutions have to initiate the use of the latest technologies and impact of E-content usage need for of the hour.

3.4 Model and Description of the Proposed Model

The proposed model describes the design and implementation of a domain-specific visual modeling tool, DL 5S CADDTIE Graph, aimed at modeling digital libraries. DL 5S CADDTIE graph is based on a metamodel that describes digital libraries using the DL 5S CADDTIE theory. The output from DL 5S CADDTIE graph is a digital library model that is an instance of the metamodel, expressed in the DL 5S CADDTIE description language (5SL). DL 5S CADDTIE graph presents the metamodel in a structured toolbox, and provides a top-down visual building environment for designers.

The visual proximity of the metamodel and instance model facilitates requirements gathering and simplifies the modeling process. Furthermore, DL 5S CADDTIE graph maintains semantic constraints specified by the DL 5S CADDTIE metamodel and enforces these constraints over the instance model to ensure semantic consistency and correctness. DL 5S CADDTIE graph enables component reuse to reduce the time and efforts of designers. The results from a pilot usability test confirm the usefulness of DL 5S CADDTIE graph.

DL 5S CADDTIE is an XML realization of the 5SL model. It is a high-level, domain-specific language. It is specific to digital libraries and represents digital libraries at a very fine granularity. If a digital library generator is to be built with the 5SL language, a semi-automatic digital library generation process would include the following steps:

- The designer of the digital library writes a DL 5S CADDTIE specification that captures the requirements for a specific digital library
- The digital library generator is fed with the DL 5S CADDTIE file and generates a digital library for the designer
The designer does not need to be an expert in software engineering or information science. The designer only needs to have a clear conceptual picture of the needed digital library and be able to transform the conceptual picture to DL 5S CADDTIE files. This greatly reduces the burden on designers, speeds up the building process, and increases the quality of the digital libraries built.

The current proposed DL 5S CADDTIE model represents DL 5S Streams, Structures, Spaces, Scenarios, and Societies. The Streams Model specifies the communication content between digital libraries and users. The Structures Model specifies how to organize information in usable ways. The Spaces Model specifies how to present information in retrievable and usable ways. The Scenarios Model specifies available information services. Finally, the Societies Model specifies how the digital library satisfies users’ demands for information.

The proposed DL 5S CADDTIE, 5S provides a formal model to capture the complexities of digital libraries. The formality of that model makes it possible to clearly specify the characteristics and behaviors of digital libraries. The formality also enables automatic mapping from DL 5S CADDTIE models to actual implementations.

3.5 Modeling of Digital Libraries

Digital libraries are usually huge and complex information systems. It is very important to have formal models and theories for such complex systems. With formal models and theories, people are able to describe, specify, and understand complex systems precisely and clearly (Halasz 1994). Most mature classes of information systems have established formal models and theories Baeza-Yates (1999). Database systems have relational models and object-oriented models Elmasri (2000).

3.6 Design Details of DL 5S CADDTIE Model

The current process of digital library generation is based on the DL 5S CADDTIE file model and is described as follows (Figure 3.2):

- First, a digital library designer conducts a DL 5S analysis of the digital library he wants to build
Second, the digital library designer describes the digital library in DL 5S.
Then, the DL 5S file is fed to the digital library generator, along with a
component pool which includes stock components that can form the new
digital library. Examples of these stock components are SAX, DOM,
routines in the MARIAN hierarchy of classes, OAI harvester, and many
more. With the component pool, the digital library generator produces a new
digital library according to the DL 5S specification.

![Diagram](image)

Figure 3.2 Building Digital Libraries with the DL 5S File

It is not easy for digital library designers to formalize a description of
digital libraries directly in DL 5S, because the designer needs to have a firm and
thorough understanding of the DL 5S. Besides that, DL 5S has five sub-models,
where each sub-model has its own syntax and semantics, and it is hard for ordinary
designers to write completely correct DL 5S files for all parts of the model.

There are also semantic constraints between sub-models. Designers need
extra effort to keep the consistency of the whole model. Our solution is to insert an
extra step (i.e., a helping tool, DL 5S file) between the digital library designer and
the digital library generator. DL5S file provides an easy-to-use graphical interface
and automatically generates desired DL 5SL files for the designer. With the
modeling tool, the process of automatic digital library generation becomes as
follows (Figure 3.3).
A digital library expert creates a metamodel for digital libraries and feeds the metamodel to the modeling tool. The metamodel is based on the DL 5S theory. It describes a generic digital library.

The modeling tool processes the metamodel, allowing a user to visualize the components of the metamodel.

A digital library designer uses the modeling tool to describe his own digital library. The visualization of the metamodel helps the designer understand the structure of a generic digital library and reduces the learning time. The designer then can use those visualized components of the metamodel to put together the final model of his own digital library.

After the designer describes his own digital library visually, the modeling tool can transform the visualized digital library into DL 5S files that follow XML syntax.

The generated DL 5S file is fed into the digital library generator with the component pool. Then, a working instance of the desired digital library is produced.

![Diagram of the digital library generation process]

Figure 3.3 Digital Library Generation Process with 5S DL Graph

With the help of a visual modeling tool (i.e., the DL graph), the difficulty of a designer's work is minimized dramatically. The designer only needs to deal with a graphical interface and pull visual components together. It is not required for him to memorize the details of the syntax and semantics of DL 5S. Cognitive load, typing effort and typing errors are reduced.
The proposed DL 5S CADTIE model (Figure 1.5) given in Chapter 1 5S represents streams, structures, spaces, scenarios, and societies, all the 5S process streams model specifies the communication content between digital libraries and users. The structures model specifies how to organize information in usable ways. The spaces model specifies how to present information in retrievable and usable ways. The scenarios model specifies available information services. Finally, the society’s model specifies how the digital library satisfies users’ demands for information. The visualization of the model also provides guidance while the designer is building his model. Moreover, correctness and consistence can be automatically guaranteed by DL 5S Graph.

The DL 5S model describes digital libraries from five different and complementary perspectives. Different S models have different primitives and serve for different objectives. Those S models are not necessarily independent from each other. A primitive in an S model may be dependent on a primitive in another S model. Figure 3.4 gives an overview of the DL 5S model.

Figure 3.4 Overview of the DL 5S Model
3.6.1 XML Modeling Tools

There are two categories of tools for building XML files. The first
category includes an ordinary XML editing tool. The ordinary XML editing tools
are pure text editors with some particular attributes for building XML files. Those
XML editors, which are only a text editor with tree views, are included in the first
category.

The second category includes an XML modeling tool. Most XML
modeling tools provide a visual interface for users. XML modeling tools are not
modeling tools like UML modeling tools that enable users to build a model using
UML. They are also different from XML schema design tools that provide means to
build or edit an XML schema. The common work procedure of a XML modeling
tool is that it loads a DTD or XML schema, and uses the structure information to
help a user build an XML instance of that DTD or XML schema. But the actual text
may not be available for users to edit directly Speed Legal (2002)\(^6\).

SpeedLegal (2002)\(^6\) is a representative open-source example of XML
modeling tools. Xerlin arose from an open-source project. A DTD is required in
order for Xerlin to work. Xerlin provides the interface for users to add, remove,
rearrange, and edit XML elements and XML attributes. The DTD specifies what
elements are valid, and what attributes are appropriate. Currently, Xerlin does not
support XML schema.

It is observed that a tree structure and a set of semantic constraints are
enough to describe the relationships among major components of the DL 5S model.
Trees are used to represent the metamodel and the instance model in DL 5S
CADDIE MODEL. This model representation is not chosen because it adds
unnecessary layout complexity into the tool. Another reason for choosing a tree is
that both the metamodel and model are specified in XML. Tree structures in the tool
are harmonious with XML, as research shows tree structures are good at showing
XML instances Pietriga (2001)\(^7\).

3.6.2 Toolbox

The first design question to ask concerns how the tool helps the user
build a model that is an instance of a metamodel. A component in the metamodel
represents a type, and a component in the instance model represents an instance of a certain type. The requirement behind the question is that any component that appears in the instance model must be an instance of a certain type in the metamodel. This requirement has two implications. First, a component instance in the instance model should have all the properties of the corresponding type of the metamodel. Second, the component instances in the instance model should maintain the same structural relationships as their corresponding types in the metamodel.

Some visual systems, e.g., for Concept Maps (IHMC01), do not have predefined visual components, or predefined structural relationships. They utilize generic visual components and links. In a Concept Map, those generic visual components with different labels can be used to represent different concepts. Thus, links with different labels stand for different relationships. Figure 3.5 represents the example of a screen shot of a concept map. The major disadvantage of applying this approach to digital library design is that, without predefined types and relationships, the user would have to build the components from scratch and maintain correct structural relationships among them. This process is usually very time-consuming and tedious. It is particularly true when the metamodel is large and complicated.

Figure 3.5 A Screen Shot of a Concept Map
Another disadvantage of this approach is that the system would need a very powerful type-checking mechanism to ensure that all the types, attributes, and relationships are included correctly. Therefore, the majority of modeling tools do not adopt this approach. Instead they prefer to use predefined components and relationships.

Most visual systems, which have predefined components, use a toolbox to show all available visual components. The visual components can be chosen and added to a user’s model directly. Example systems are the BeanBox for JavaBeans Sun (2002a)\textsuperscript{8}, Khoral (2001)\textsuperscript{9}, and many other commercial visual component software packages. Most toolboxes are in the form of either a list, as in the BeanBox, or a categorized list. A typical toolbox is shown in Figure 3.6 Sun (2002b)\textsuperscript{10}. The relationships between those components obey the rules defined by a certain metamodel. Only valid connections are allowed. The problem with these toolboxes is that the structural relationships among the visual components are not visible from the toolbox. The user needs to have sound knowledge of the metamodel in order to maintain the correct structural relationship among components. A trial-and-error approach is necessary to build a correct system.

![Figure 3.6 The Toolbox of BeanBox](image)

To overcome the disadvantages of unstructured toolboxes a structured toolbox is introduced into the proposed tool, which not only provides all the visual
components of the metamodel, but also shows the structural relationships among these components (Figure 3.7).

Figure 3.7 The Workspace and the Structured Toolbox in DL 5S CADDTIE Model

As illustrated by Figure 3.13, the tool is divided into two parts. The lower part is the structured toolbox that shows all available components of the metamodel and the relationships among them. It also can be considered as a visualization of the metamodel. The upper part is the workspace in which users can create their instance models.

3.6.3 Icons

Visual components in the toolbox and in the workspace have different meanings and different behaviors. However, they all have the same shape (a rectangle). The reason for using the same shape for all visual components is that the tool itself does not have any predefined shape/type association. The tool does not know a prior what types it will have before it gets the input of a certain metamodel. Using the same shape for all components, the layout strategy is simplified and consistent visualization effects are obtained. In fact, many visualization systems use the same shape (usually a rectangle) for all components.

However, there is an issue related with this. A visual component in the toolbox represents a type. A visual component in the workspace represents an instance. For each type, there could be several instances. Each visual component has
a name to distinguish itself from others. The names of visual components in the toolbox are not alterable, since they are type names. The names of visual components in the workspace are changeable, since they are instance names. As a result, the user may easily forget the association of an instance and its type after the instance name is changed. This is illustrated in Figure 3.8.

![Figure 3.8 Visual Components without Icons and Cardinality](image)

To solve the problem, the types are associated with different icons, which are inserted at the left side of the rectangle of every component. The names of the instances may change, but the icons do not change, which helps the user to find the correct type for their instances, as shown in Figure 3.9.

![Figure 3.9 Visual Components with Icons and Cardinality](image)
Another difference between the component in the toolbox and that in the workspace is that the component in the toolbox has cardinality associated with it, denoting how many instances of this component can be added to a parent node in the workspace. The two occurrence cases are distinguished specifying whether an instance of the type should occur at most once, or any number of times. The indicator (•) means any number of times. Having no indicator means it can occur at most once. More cardinality indicators will be added as necessary in the future.

### 3.6.4 Tree Representation

The analysis of the DL 5S CADDTIE model shows that the structure of the DL 5S metamodel can be viewed as a tree structure, accompanied by a set of domain specific constraints. Next two different approaches are compared to representing tree structures and give reasons why the Node-Link approach is chosen.

### 3.6.5 Node and Link Approach

The traditional way to represent a tree structure is with a node and link representation, which is a rooted, directed graph, as in Figure 3.11(a). There are some existing integrated Development Environments that provide graphic representation of tree structures, like Java’s JTrees Sun (2002b)\(^{10}\), illustrated in Figure 3.10. The traditional approach has its advantages and disadvantages.

The advantages of the node-link approach include:

- It is straightforward.
- The absolute depth can be easily observed, since the nodes at the same level have the same depth.
- It is easy to manipulate.

The disadvantages of the node-link approach include:

- It wastes most of the occupied screen space.
- It does not scale well. A scroll bar is needed for large structures. For example, in JTree, a deep node may cause too much vertical expansion.
To overcome visualization problems associated with traditional trees, researchers invented several variants, e.g., the hyperbolic tree Pirolli(2001)\(^{11}\) and the cone tree Robertson(1993)\(^{12}\). These variants improve visualization capabilities, with the cost of reducing the convenience of manipulation.

### 3.6.6 Nested Approach

Treemap Shneiderman (2001)\(^{13}\), employing a nested display method, is a totally different approach from the node-link representation. Trees are shown with a 2-D space-filling representation, which consists of nodes as rectangles and nested child nodes in their parent (Figure 3.11).

![Node-Link Representation](image)

![Treemap Representation](image)

a). Node-Link Representation.  

b) Treemap Representation

Figure 3.11 Node-Link and Treemap Representations of a Tree
The advantages of Treemap include:

- better use of the screen real estate, and the
- ability to visualize large hierarchies.

The disadvantages include:

- It requires zooming capability, because the deep nodes may be very small.
- No obvious information about depth is available.

3.7 Reasons for Choosing Node-link Representation

The traditional node-link representation is chosen for this work because the depth information is obvious, while a nested representation does not convey the depth information as clearly. The node-link representation shown in Figure 3.12 makes clear the correspondence between the levels of the metamodel and the user's instance model.

![Node-link Representation with Clear Depth Information](image)

It is required that the level of an instance component in its instance tree should be the same as the level of its corresponding type component in the metamodel tree. Since it is very important for the tool designer to have an explicit and clear overview of the depth information of both trees, the node-link representation is preferred.
3.8 Truncated Visualization of Full Trees in DL 5S CADDTIE Model

There often will be many nodes at deep levels of a large tree. Theoretically, for an n-ary tree, there are at most nh−1 nodes at level h. For example, there are at most 81 nodes at level 5 of a ternary tree. The number of nodes being displayed are to be reduced to avoid serious difficulties with layout and icon search speed Byrne (1993). A simple but very efficient solution is to show only one branch, which is in the form of a list, at each level. An example is shown in Figure 3.13.

Figure 3.13 A Tree with One Visible Branch at Each Level
The parenthetical after the name of a node gives the number of direct child nodes.

3.9 Visible Path and Consistent Lists

For the convenience of explaining the truncated display and the user interactions, we introduce the concepts of visible path and consistent lists. The path from root to the deepest visible leaf is referred to as a visible path. The path from any visible node to a deepest visible leaf is referred to as a visible path starting from that visible node. A visible path is:

A visible path starting from Struct_Model is:

A node list, ListA, in the toolbox and a node list, ListB, in the instance model are consistent with each other if and only if the parent node of ListB has the
type of the parent node of ListA. As a result, all the types of the components in ListB are in ListA.

An example is shown in Figure 3.14. The two-way arrows point to lists that are consistent with each other. The list of S models in the user model ("Your digital library") is consistent with the list of S models in the metamodel ("Digital library model"). The list of streams is consistent with the list of stream types in the metamodel.

![Figure 3.14 Consistent Lists](image)

### 3.10 Truncated Display

Truncated display of trees in DL 5S CADDIE minimizes layout difficulties, reduces the size of the icon set that needs to be searched, and focuses a user's attention on the currently active branch. Although a truncated display shows less information than a full display, it shows the most relevant information in the tree. It manages to dynamically focus only on the active branch of the tree on which the user is working, while leaving out irrelevant branches.
The information presented in the truncated display includes the:

- parent node of a node,
- child nodes of a node
- siblings of a node and
- level of the node in the entire tree.

For example, in Figure 3.20, the information that is displayed for the chosen component 
Stream_Model includes:

- Its parent component: Digital_Library.

The level of Stream_Model in the metamodel tree is 2.

3.11 User Interactions

In this section, we first describe some basic operations supported by DL 5S CADDTIE Model are described. Then an important feature of DL 5S CADDTIE Model: the synchronization mechanism is explained. Finally component reuse in DL 5S CADDTIE Model is discussed.

3.11.1 Basic Operations

DL 5S CADDTIE Model supports four basic operations: exploring, adding, deleting, and changing properties. They are introduced and described in detail in the following paragraphs.

3.11.2 Exploring

Exploring is invoked by single clicks. The operation is similar to the expanding operation of JTree in Java. It enables the subtree of a selected node to be visible.
3.11.3 Scenario of Exploring

A user clicks on the node of Stream_Model and looks at the content of Stream_Model that is included in the subtree (Figure 3.15). After that, the user clicks on the node of Struct_Model and checks the content of the Struct_Model that is now visible in the subtree of Struct_Model. Struct_Model has three children: CollectionSet, CatalogSet, and Organization_tools. The user is interested in CollectionSet, so he clicks on the CollectionSet and looks at the children of it. In this way, the user can see the entire model (Figure 3.16).

Figure 3.15 Operation of Exploring

Figure 3.16 Operation of Exploring (Entire Model)
3.11.4 Usability Tips of Exploring

The tool keeps track of all visible paths starting from any node. As such, if a user clicks on any previously selected node, the tool can immediately show to the user the last visible path starting from that node. This design is based on the assumption that the path that a user has just visited is most likely to be visited again.

3.12 Adding

The user adds an instance into the instance model by double clicking on a component in the toolbox. Alternatively, the user may right click on a component in the toolbox and choose the adding operation from the pop-up menu.

DL 5S CADDTIE Model utilizes a top-down modeling strategy in creating a model. The user has to add the parent into the model before he/she can add any child components. If a new child component is to be added into the model, just like when using any similar tools, the user needs to specify a position for the new component. This is done by selecting a parent node first and then appending the child component to the end of the children list of the parent node.

3.12.1 Scenario of Adding

A user browses through his model. When he reaches the Stream Model, his click on Stream_Model brings up all the child nodes. He finds out that he needs to add one more image type, image/gif, to his model. So he double-clicks on the image/gif component in the toolbox and adds it to his model. A top-down assumption: the user clicks on the parent node to check its content before he decides to add a component to the parent node (Figure 3.17).

![Figure 3.17 Scenario of Adding](image)
3.12.2 Usability Tips of Adding

When the user selects a component in his instance model, the tool synchronizes with it by showing a visible path in the toolbox that goes from the root to the type of the selected component. Therefore, the user always knows which components in the toolbox can be added as the sibling or children of the selected component.

3.12.3 Deleting

Pressing the “DEL” key triggers deletion. The user can only delete nodes in the user model.

3.13 Changing Properties

Each instance component has its own property sheet that goes with its type. The property sheet is not visualized in the toolbox, or in the workspace, because it adds unnecessary complexity. The user can get the property sheet by double clicking on the component in the user model. The properties in the property sheet are changeable (Figure 3.18).

![Figure 3.18 The Operation of Changing Properties](image)

3.14 Synchronization

There are two views in the tool. One is for the toolbox (metamodel); the other one is for the user model. These two views are related through the
type/instance relationships between components in the toolbox and components in the user model.

When a user selects an instance component in the workspace (user model), DL 5S CADD TIE Model is able to synchronize the view of the toolbox by showing a visible path from the root to the corresponding type component of the selected instance component. The convenience of synchronization is that:

- The user does not need to manually search all the components in the toolbox to find the correct type component
- The tool helps the user focus on the most important relationships of the type Components
- The child components that can be added to the current component are within immediate reach of the user

3.15 Scenario of Synchronization

The user clicks on “MyStreamModel”. The toolbox finds that the type of the component is Stream_Model, and expands the subtree of Stream_Model automatically. The user adds an instance of Audio to MyStreamModel (Figure 3.19). The user then clicks on MyStructModel. The toolbox expands the subtree of Struct_Model automatically. The user looks at the child nodes of Struct_Model and adds an instance of CollectionSet and an instance of CatalogSet to his model (Figure 3.20).

![Figure 3.19 The User Adds an Instance of Stream_Model](image)
3.16 Component Reusability

In the proposed case, component reusability means the components that have been built in one user model can be saved and reused by other user models. Reusability saves time and effort. There are components that are common for many different digital library systems. For example, many digital libraries share the same data formats, and the same descriptive metadata. The components representing the Stream Model or the metadata can be built once and reused in different digital libraries.

When a new component is needed, the user does not need to build a component from scratch. He loads a similar component and spends relatively less time by making minor changes to the loaded component.

3.17 Scenario of Component Reuse

A user builds the Stream Model once and saves it as a file on a local disk. Later, when the user starts to build a new digital library, he right-clicks on the new digital library name, chooses "load" in the pop-up menu (Figure 3.21) and loads the previously saved Stream Model file. Then, a Stream Model appears in the new digital library. Figure 3.22 shows the stream model before loading.
3.18 Non-Reusable Components

Not all components are reusable. A component to be reused should not contain any parts that are constrained to a specific model. In other words, a reusable component should be self-contained and independent of any other specific models.

3.19 Top-down or Bottom-up Methodology

DL 5S CADDTIE Model is designed to support a top-down modeling methodology. With the aid of component reuse, the tool also can be used in a bottom-up way. The essential steps are as follows.

- Create a metamodel for each type of component
- Use the tool to build basic components as building blocks
- Build higher-level components after all building blocks have been built
Build top-level component after all underlying components have been built

The bottom-up methodology relies on the fact that DL 5S CADD TIE model can create not only a complete digital library, but also the components of a digital library. For example, an instance of the Stream Model using DL 5S CADD TIE model can be created (Figure 3.23). In this example, a metamodel for the Stream Model is loaded first. Then, a Stream Model instance is created which later can be reused to build a complete digital library. This illustrates the bottom-up methodology.

![Figure 3.23 Create an Instance of the Stream Model Using DL 5S CADD TIE Model](image)

3.20 Objectives

In the proposed work the idea that visualization helps people understand complex models is adopted. DL 5S Graph is able to load and display digital library meta models. The designer does not need to memorize all the details of digital library modeling theories. The visual model shows the structure and different concepts of a digital library and the relationship among these concepts. DL5SGraph also provides a structure editor to let the designer build a digital library by manipulation and composition of visual components. Furthermore, DL5SGraph is able to produce correct DL 5SL XML files according to the visual model built by
the designer. Syntactical details of the DL 5S model and DL 5 SL are hidden from
the designer. As such, DL 5S Graph eliminates the disadvantages of DL 5 SL.
The study has been carried out with the following objectives.

✓ To describe properties of the digital library content such as encoding and
language for textual material or particular forms of multimedia data
✓ To specify organizational aspects of the digital library contents (e.g.,
structural /descriptive metadata, hypertexts, taxonomies, classification
schemes)
✓ To define logical and presentation views of several components.

The more specific objectives of the proposed work are

✓ To help digital library designers understand the DL 5S CADDITIE
model quickly and easily
✓ To help digital library designers build their own digital libraries without
difficulty
✓ To help digital library designers transform their models into DL 5S
CADDITIE files automatically
✓ To help digital library designers understand, maintain, and upgrade
existing digital library models conveniently

3.21 Datatypes and Datasets

A metamodel is central to the functioning of the DL 5S CADDITIE
Model tool. Without a metamodel, the tool is just an empty facility for hierarchical
frameworks with no substance. A metamodel describes a generic digital library,
provides the building blocks, and sets up the relationships among these building
blocks. A metamodel is based on the DL 5S theory and should be created by a
digital library expert who has a good understanding of the DL 5S theory.

As we point out in previous chapters, input from digital library experts
to the DL 5S CADDITIE model tool includes not only modeling information, but
also configuration information for visual effects, such as icon/type associations. This
configuration information also is included in the specification of the proposed
metamodel.
To summarize, a metamodel should provide the following information:

- Component types
- Properties with each type
- Icon/type associations
- Hierarchical structure information
- Cardinality constraints
- Semantic constraints

3.21.1 DataSet

The metamodel of DL5S CADTIE specifies two types of elements: DataType and DataSet. The root element is always DLMetaModel. All DataSet elements must be defined before the elements of DataType are given.

The structure attributes, and visual notations for dataset elements are explained in the following sections. Scenario examples also are given to facilitate better understanding of these concepts.

DataSet represents a set of data. It is a range of data from which a value can be chosen. For example, a declaration for MetaData is illustrated as follows:

```
<MetaData_Content type="DataSet">
  <item value="Dublin Core" />
  <item value="IMS" />
  <item value="MARC" />
  <item value="RFC1807" />
  <item value="IEEE LTSC-LOM" />
</MetaData_Content>
```

The attribute type demonstrates that MetaData_Content is a DataSet, which has a set of values. The above example means MetaData_Content has values of Dublin Core, IMS, MARC, RFC1807, and IEEE LTSC-LOM. DataSet has no independent visualization in the tool. It only acts as a data source that is used by a DataType. A DataSet may represent a set of image types, a set of audio types, and many other value sets details as,

- <Image_Content type="DataSet">
  <item value="image/gif" />
```
3.21.2 DataType

A DataType is represented by a component of the metamodel, which specifies a type and can be instantiated into many instances in the user model. Examples of DataTypes are digital library, stream model, structure model, and document. An element that has been declared as a DataType is visualized in the toolbox as a named rectangle. Every DataType element consists of three parts: SubNodes, Icon, and Property. In the following sections, the visualization of each part is introduced, along with examples and scenarios.

3.21.3 SubNodes and Icon

An example of a DataType declaration and its visual notation appears in Figure 3.24(a) and 3.24(b).

![Digital Library and its Visual Notation](image/dl.gif)
The attribute type shows that the element Digital_Library is aDataType.
The element SubNodes contains other DataTypes that are the child nodes of this
element. In this example, the SubNodes of Digital_Library consist of
Stream_Model, Struct_Model, Space_Model, Scenario_Model, and Society_Model.
In the visualization, the number of child nodes is specified in parenthesis after the
name of Digital_Library. The Icon specifies that the image file “dl.gif” is used as the
icon for the element Digital_Library. No Property is defined for Digital_Library.

The Digital_Library example does not specify the cardinality of the child
nodes, which means the default cardinality is used. The default value of cardinality
is set to be “at most once”. Therefore, Digital_Library can have at most one
Stream_Model, at most one Struct_Model, at most one Space_Model, at most one
Scenario_Model, and at most one Society_Model. However in the Stream_Model,
its cardinality is explicitly specified.

```
<Stream_Model type="DataType">
  <SubNodes>
    <Audio constraint="*" />
    <Video constraint="*" />
    <Text constraint="*" />
    <Image constraint="*" />
    <Application constraint="*" />
  </SubNodes>
  <Icon name="strM.gif" />
  <Property />
</Stream_Model>
```

Figure 3.24 (b) Stream_Model and its Visual Notation

The SubNodes part contains four other DataTypes: Audio, Video, Text,
Image and Application. This means that Stream_Model has five types of elements as
its child nodes. The “constraint” attribute specifies the cardinality. The constraint of
‘*’ means ‘any’ which means an instance of the Stream_Model may have any
number of instances from among the kinds: audio, video, text, etc.

**Scenario 1**

Purpose: This scenario shows how to use an element of DataType in the user
model. A user wants to create his own digital library (Figure 3.25).
Figure 3.25 Screen Shot for Scenario 1

First, he adds an instance of Digital Library to his model. Then he clicks on the instance and changes the name to My_DL. His clicking on the instance triggers the synchronization mechanism. The tool expands the Digital Library and shows the child nodes of Digital Library in the toolbox. Now, the user can add the necessary child nodes to his model. He adds a Stream_Model and a Struct_Model, and changes the names to My Streams and My Structures.

Scenario 2

Purpose: This scenario shows how to use cardinality in DataType.

The user (Figure 3.26) decides to specify streams in his Stream_Model: My_Streame. When he clicks on My_Sstreams, the tool synchronizes to show the child nodes of the Stream_Model in the toolbox. Therefore, he knows what kind of streams his Stream_Model can have. He adds three kinds of audio: Basic_Audio, Wav_Audio, and Aiff_Audio.

There are three points we need to pay attention to:

✓ The user can add any number of audio streams because the cardinality of audio in Stream_Model indicates the instance of Audio can appear any number of times with one parent
✓ Cardinality is only indicated in the toolbox (i.e., the metamodel), not in the user model
✓ The icons of all the instances of Audio are the same, because they are of the same type (audio type)
3.21.4 Property

The Property of a DataType specifies what properties the element should have. All the specified properties are shown on the property sheet that is associated with each DataType element. The metamodel may specify properties for a DataType element or may not specify any properties.

If no property is specified for an element, the element will only have the default properties on its property sheet. For example, no property is specified for the element Digital_Library. All properties that the Digital_Library can have are those default properties. The property sheet of Digital_Library is shown in Figure 3.27. The ‘Node Type’ and ‘Enter a new name’ are default properties that every DataType element will have.

```
<Digital_Library type="DataType">
  <SubNodes>
    <Stream_Model />
    <Struct_Model />
    <Space_Model />
    <Scenario_Model />
    <Society_Model />
  </SubNodes>
  <Icon name="dl.gif" />
  <Property />
</Digital_Library>
```

Figure 3.27 Digital_Library and its Property Sheet
The properties that can be specified in Property are in the format of interface elements: Combo Box and Text Field. The next example shows how to use Combo Box in the Property part. The declaration of DataType Audio (Figure 3.28) is:

```xml
<Audio
    <Icon />
    <Property>
        <ComboBox name="content-type" src="Audio_Content" />
    </Property>
</Audio>
```

![Figure 3.28 Audio and its Property Sheet](image)

The property sheet of Audio has three parts: Node Type, Enter a new name, and a ComboBox named "content-type". The content of the ComboBox is obtained from the DataSet "Audio_Content". Text field is another interface component that can be added to the property part. For example (Figure 3.29). DataType Catalog has four defined properties as shown below.

- `<Catalog type="Data Type"/>
- `<SubNodes>`
  `<MetaDataFormat constraint="*" />`
- `<Icon />
- `<Property>`
<TextField name="Description" />
<TextField name="Creator" />
<TextField name="Maintainer" />
<ComboBox name="Collection" src="CollectionSet" />
</Property>
</Catalog>

Figure 3.29 Catalog and its Property Sheet

Three text fields “Description”, “Creator”, and “Maintainer” are added to the property sheet of Catalog. If the text field has an attribute ‘Load’ with the value of ‘true’, it means the value of the property can be loaded from some file on disk. Figure 3.30 shows the ontology and its property sheet.

```xml
<Ontology type="DataType">
  <SubNodes />
  <Icon />
  <Property>
    <TextField name="Load_Ontology" load="true" />
  </Property>
</Ontology>
```

Figure 3.30 Ontology and its Property Sheet

The specification and the visualization of the metamodel is also described. In the following section, a very important feature of the DL 5S CADDIE Model tool is introduced – constraint management.
3.22 Constraint Management

The analysis of DL 5S theory that there are certain inherent semantic constraints in the hierarchical structure of the DL 5S CADDTIE model. The semantic constraints in DL 5S are divided into two categories: value constraint and association constraint. The value constraint specifies the range of possible values of an element, while the association constraint defines the relationships among different components. For example:

✓ An Actor can only participate in the services that have been defined in the Scenario model. This is a value constraint on the services
✓ A Catalog must have a 1:1 relationship with a Collection. This is an association constraint between a Catalog and a Collection.

The DL 5S CADDTIE Model tool is able to implement and manage these constraints. The following sections explain how the metamodel specifies constraints and the modeling scenarios.

3.22.1 Value Constraint

Example: An Actor can only participate in the services that have been defined in the Scenario Model. The declaration of Actor details are

```xml
  <Services type="DataType">
    <SubNodes>
      <Scenario constraint="*" />
    </SubNodes>
    <Icon name="strM.gif" />
    <Property>
      <TextField name="Load_Services" load="true" />
    </Property>
  </Services>

  <Actor type="DataType">
    <SubNodes src="Services" />
    <Icon />
    <Property />
  </Actor>
```

The SubNodes part of Actor specifies that the child nodes of Actor come from source 'Services', which means only the existing instances of Services can become the child nodes of Actor.
Scenario 3

Purpose: Shows how DL 5S CADDTIE Model maintains value constraints.

The designer is building a model for the NDLTD digital library using DL 5S CADDTIE Model (Figure 3.31). He browses to the Society Model and adds anew an Actor named “Students” to his instance model. If he wants to add services to students now, he cannot do it because no service has been specified in the Scenario Model yet. (This maintains the value constraint.)

![Figure 3.31 Screen Shot 1 for Scenario 3](image)

Then, the designer browses to the Scenario Model part and creates five services: fulltext_search, metadata_search, browsing, submission, and training (Figure 3.38).

![Figure 3.32 Screen Shot 2 for Scenario 3](image)

When the designer browses back to Students, he finds out that five services are automatically added into the metamodel under the node “Actor” (Figure 3.33). 5SGraph dynamically maintains this for the user.
The designer then adds all five services to Students (Figure 3.34), which means Students can participate in all five services provided by the NDLTD digital library.

3.22.2 Association Constraint

Example: A Catalog has descriptive metadata for digital objects in a specific Collection. Therefore, a Catalog must have a 1:1 relationship with a Collection, which means a Catalog is not independent. A Catalog must depend on one existing Collection.
The declaration of Catalog is described below:
- <Catalog type="DataType">
- <SubNodes>
  <MetaDataFormat constraint="*" />
</SubNodes>
- <Icon />
- <Property>
  <TextField name="Description" />
  <TextField name="Creator" />
  <TextField name="Maintainer" />
  <ComboBox name="Collection" src="CollectionSet" />
</Property>
</Catalog>

In the declaration of Combo box, the attribute src refers to CollectionSet, which is a DataType, not a DataSet. If the src refers to a DataSet, the content of a Combo box is fixed, because the data from the DataSet is fixed. If the src refers to a DataType, the content of the Combo box depends on the content of the instance of that DataType. The content of a Combo box dynamically changes with the user model. In this case, the content of the ComboBox depends on what the designer puts into the CollectionSet.

**Scenario 4**

Purpose: Illustrate how to utilize an association constraint.

A designer is building a model for the CITIDEL digital library. He should first build nine collections in his Structural Model (Figure 3.35). Then he should create a catalog (Figure 3.36).

In the property sheet of the Catalog, the ComboBox forces him to choose a collection from existing collections for this catalog. He chooses the Virtual_Union_Collection and then names the Catalog as Union_Catalog.
From the above examples, it is concluded that it is very easy and convenient to maintain and manage semantic constraints (e.g., a value constraint and association constraint) using DL 5S CADDTIE Model. This feature of DL 5S CADDTIE Model greatly improves the usability of the tool and frees the user from the burden of remembering and maintaining the constraints manually to content in digital library.
REFERENCES


URL: http://java.sun.com/j2se/1.3/docs/api/.

