Chapter 7

Pragmatics: Tools for a Multimedia Development Environment

In the preceding four chapters we described the syntax of multidimensional languages to specify the presentation of multimedia applications, and the semantics of the teleaction objects to specify the activities performed by multimedia applications. In this chapter we give an overview to a software engineering environment referred to as the Multimedia IC Development Environment (MICE), and its associated tools. The details of the MICE tools will be presented in Chapter 8.

MICE is to be used as the basis for the study of the visual design process applied to the development of TAO-based multimedia applications. The unifying model used in this approach is based on Teleaction Objects (TAOs). TAOs are multimedia objects with attached knowledge structured as an active index. TAOs can be described using the TAOML extension of HTML. This allows for easy prototyping of distributed multimedia applications using a web browser as the user interface. The TAOML Builder tool allows the user to visually specify a TAO. The hypergraph is parsed for correctness using an underlying Boundary Symbol Relation grammar and the correct TAOML is output. TAOML can be translated into standard HTML using the TAOML Interpreter. The ICs for the application can be visually specified using the IC Builder. The IC Compiler produces the IC Manager that provides the run-time environment for the ICs.

1. THE MICE ENVIRONMENT

Distributed multimedia applications have become increasingly common in recent years due to the development of the World Wide Web. Unfortunately, supporting tools and techniques for such applications are not readily available. The goal of this research is to study the visual software design process applied to multimedia applications by developing a visual software engineering environment [Costa95a] for such applications. In previous chapters, we have described the formal framework that can be used as the basis for application development. Based on this approach, a set of tools for the production of multimedia applications has been developed at the University of Pittsburgh and the University of Salerno. These tools are based on the Teleaction Object (TAO) paradigm. TAOs are multimedia objects with attached knowledge in the form of a collection of index cells (ICs) comprising an active index [Chang95a]. The set of tools comprising the workbench is referred to as the Multimedia IC Development Environment (MICE). In the MICE approach to TAO-based multimedia application development, TAOs are described using TAOML, an extension of HTML. This allows for easy prototyping of distributed multimedia applications using a standard web browser as the user interface. The tools comprising MICE are: TAOML Builder; TAOML Interpreter; IC Builder; IC Compiler and IC Manager. The Interactions of these tools are shown in Figure 1.

The MICE approach is especially suited for quickly prototyping complicated distributed multimedia applications, including those interacting with database management systems. The use of a visual software engineering environment helps to manage the *structural complexity* [Karsa95] of such applications. Due to the fact that the approach uses a standard web browser as the user interface, as well as the implementation of the IC Manager in standard C language, the developed application may be easily ported to the desired environment.

The rest of the chapter is structured as follows. Section 2 contains a brief review of TAO-based multimedia application development. Sections 3 through 8 contain descriptions of each of the MICE tools.

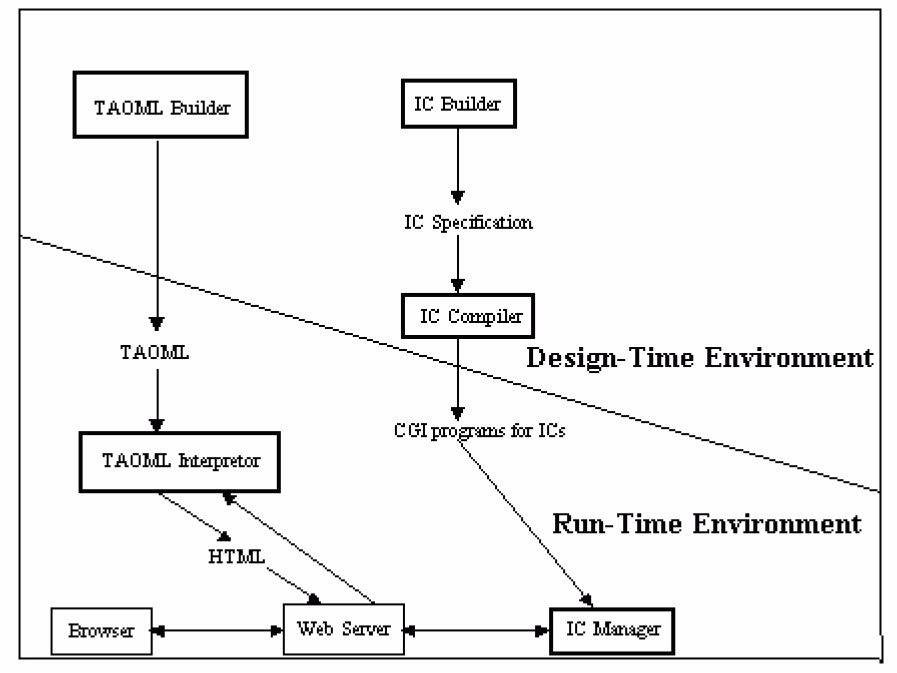


Figure 1. MICE Tools.

2. TAO-BASED MULTIMEDIA APPLICATIONS

Teleaction Objects (TAOs) are multimedia objects with an associated hypergraph representing the structure of the multimedia object and a knowledge structure. The knowledge structure allows the TAO to automatically react to certain events [ChangH95b].

From a structural point of view, a TAO can be divided into two parts: a *hypergraph G* and *knowledge K*.

The structure of the hypergraph G is a graph G(N,L), where N is a set of nodes, and L is a set of links. There are two types of nodes: base nodes and composite nodes. Each node represents a TAO, and each link represents a relation among TAOs and there are the following link types: the *attachment link*, the *annotation link*, the *reference link*, the *location link*, and the *synchronization link*. Base nodes and composite nodes are called *bundled* when they are grouped, thus defining them as a single entity. The nodes which are interior to bundled nodes may not be included in annotation or reference links unless the link is to the exterior bundled node, and there may not be spatial/temporal relations between interior nodes and nodes external to the bundled node.

The knowledge structure K of a TAO is classified in four levels: the *System Knowledge*, the *Environment Knowledge*, the *Template Knowledge*, and the *Private Knowledge*. The knowledge is structured as an *active index* (IX), which is a set of *index cells* (IC) from an *index cell base* (ICB). The index cells define the reactions of the TAO to events filtered by the system. An index cell accepts input messages, performs some action, and sends output messages to a group of ICs. The messages sent will depend on the state of the IC and on the input messages [Chang96a]. An IC may be seen as a kind of finite-state machine [Chang95a].

An initial approach to the definition of a multimedia language for TAOs has been given in [Chang96a]. The physical appearance of a TAO is described by a *multidimensional sentence*. The language is generated by a grammar whose alphabet contains generalized icons and operators. Formally, a generalized icon is defined as $x=(x_m,x_i)$ where x_m is the meaning of the icon and x_i is the media object. Two functions, materialization and dematerialization, are associated with every generalized icon. The first function derives the object from its meaning: $MAT(x_m)=x_i$; the second derives the meaning, or interpretation, from the object: $DMA(x_i)=x_m$.

The generalized icons [Chang87b] are divided into the following categories:

• Icon : (x_m, x_i), where x_i is an image

• Earcon : (x_m, x_e), where x_e is a sound

• Ticon : (x_m, x_t), where x_t is text (the ticon can also be seen as a subtype of icon).

• Micon : (x_m, x_s), where x_s is a sequence of image icons (motion icon)

• Vicon : (x_m, x_v), where x_v is a video clip (video icon)

• Multicon : (x_m, x_c), where x_c is a multimedia sentence (composite icon).

The generalized icons are represented by nodes in the hypergraph while operators are represented by links.

2.1 TAOML

In order to more easily prototype a distributed multimedia application based on the TAO concept, an extended version of HTML called TAOML has been developed. TAOML can be regarded as a subclass of XML. With TAOML, each component of the application can be realized as an ic associated with a TAO-enhanced html page. Given a TAO-enhanced html page, we can use an interpreter to read this page, abstract the necessary TAO data structure and generate the normal html page for the browser. Therefore no matter which browser is used, the application program can run if this TAO_HTML interpreter is installed in advance. This can give some security guarantees. The user can also choose a favorite browser. Furthermore if in the future HTML is out of fashion, the user just needs to update the interpreter and change it into another language. The other parts of application will not be affected. In this section, we describe the TAO enhanced html named TAOML.

In order to use TAO_HTML, or TAOML, to define a TAO, the data structure of a TAO is extended. A TAO has the following attributes: tao_name, tao_type, p_part, links, ics and sensitivity.

- 'tao_name' is the name of the TAO, which is a unique identifier of each TAO.
- 'tao_type' is the media type of TAO, such as image, text, audio, motion graphs, video or mixed.
- 'p_part' is the physical part of TAO. To implement it in the context of TAO_HTML, 'p_part' here can be denoted by a template that indicates how an HTML page looks.
- 'links' is the link to another TAO.
- 'ic' is the associated index cell.
- 'sensitivity' indicates whether this object is location-sensitive, time-sensitive, content-sensitive or none-sensitive. Then the same object can have different appearance or different functionality according to the sensitivity. The detailed meaning of sensitivity should be defined by user according to the requirement of applications.
- 'database' specifies the database that this TAO can access and/or manipulate.

The formal definition of TAO_HTML language can be described in BNF form:

```
TAO_HTML ::= <TAO> TAO_BODY </TAO>

TAO_BODY ::= NAME_PART TYPE_PART P_PART LINK_PART
    IC_PART SENSI_PART DATA_PART

NAME_PART ::= <TAO_NAME> "name" </TAO_NAME>

TYPE_PART ::= <TAO_TYPE> TYPE_SET </TAO_TYPE>

TYPE_SET ::= [image, text, audio, motion_graph, video, mixed]
```

```
P_PART ::= <TAO_TEMPLATE> "template_name" </TAO_TEMPLATE>

LINK_PART ::= empty | <TAO_LINKS> LINK_BODY </TAO_LINKS>
    LINK_PART

LINK_BODY ::= name = "link_name", type = LINK_TYPE, obj = "link_obj"

LINK_TYPE ::= [spatial, temporal, structural]

IC_PART ::= empty | <TAO_IC> flag=FLAG ic_type="a_string"
    ic_id_list="a_string" cgi_pgm="a_string" message_type="a_string"
    content="a_string" </TAO_IC>

FLAG ::= [old, new]

SENSI_PART ::= empty | <TAO_SENSI> SENSITIVITY </TAO_SENSI>

SENSITIVITY ::= [location, content, time]

DATA_PART ::= empty | <TAO_DATA> "database_name" </TAO_DATA>
```

In the template of a TAO, in addition to the normal HTML tags and definitions, there is a special TAO tag for link relation with other TAOs. It is defined as:

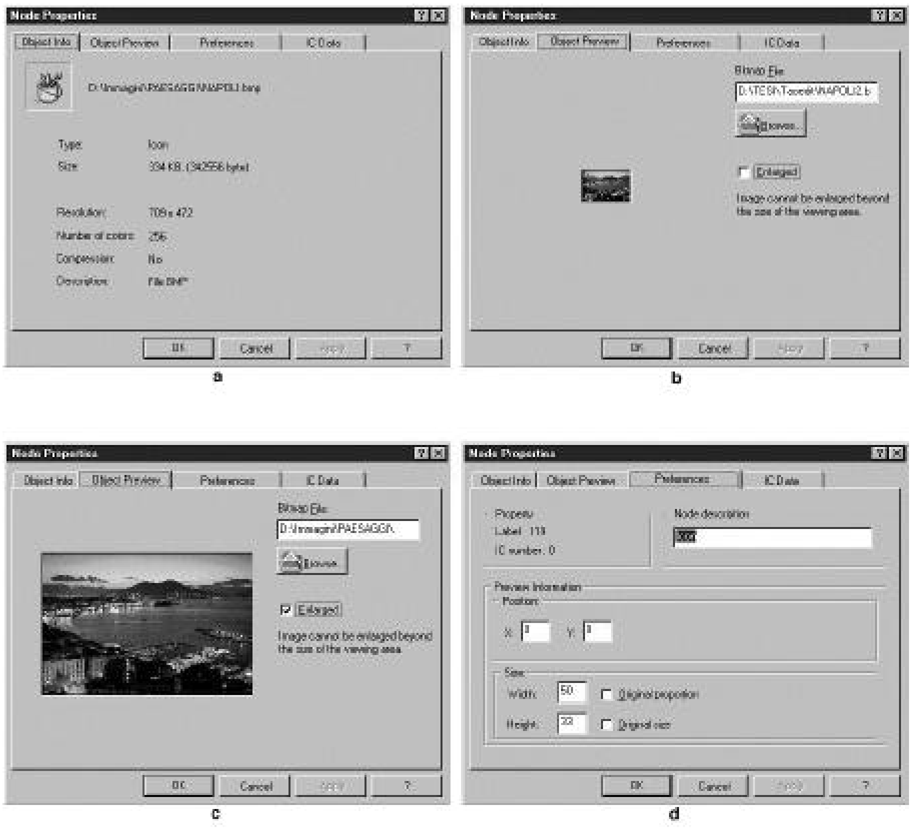
```
<TAO_REL> "link_name" </TAO_REL>
```

3. TAOML BUILDER

The TAOML Builder is a visual tool for MICE application developers. It allows users to specify the structure of a TAO in the form of a hypergraph representing the multimedia objects and the relations between these objects. Once the user has decided on the objects to be contained in a TAO and the relations to hold between the objects, the tool will automatically generate the TAOML corresponding to the visually specified TAO. This output is then used by the TAOML Interpreter tool described in Section 4. The TAOML Builder is based on an underlying multidimensional grammar (Symbol Relation grammar [Ferru96]) describing valid TAO structures [Arndt97a].

The TAOML Builder allows the creation of the nodes and links of the hypergraph of a TAO. The properties of each node of the TAO are collected in the dialog tab “Obj Info” as shown in Figure 2a. The dialog tab “Obj

Preview" gives a preview of the selected node together with some information about the file attached to the node (see Figure 2b and Figure 2c). If the node is a *micon* the component nodes of the structured icon will be listed.



D:\Immagini\PAESAGGI\NAPOLI.bmp

Type: Icon
Size: 334 KB (342556 byte)
Resolution: 709 x 472
Number of colors: 256
Compression: No
Description: File BMP

Bitmap File: D:\TESI\Tavole\NAPOLI2.b

Image cannot be enlarged beyond the size of the viewing area.

Bitmap File: D:\Immagini\PAESAGGI\

[✓ Enlarged]

Image cannot be enlarged beyond the size of the viewing area.

Property
Label: 119
IC number: 0
Node description: 300

Preview Information
Position
X: 0 Y: 0
Size
Width: 50 Original proportion
Height: 33 Original size

Figure 2. Tabbed Dialog Showing Node Properties.

The dialog tab "Preferences" allows the insertion of preferences that will influence the final presentation (Figure 2d). Information about the IC cells connected to the selected node can be inserted in the dialog tab "IC Data". A property dialog box is also provided for the links.

The tool bar of the TAOML Builder is split into four parts (Figure 3): the main tool bar which contains commands for printing, cutting, copying, etc.; the tool bar of the nodes which allows the addition as well as the removal of the nodes for the construction of the hypergraph of the TAO; the tool bar of the links which allows the addition and the removal of the links of the hypergraph. The magnifying glass allows zoom in and zoom out of the screen in order to have a complete view in one page of the hypergraph.

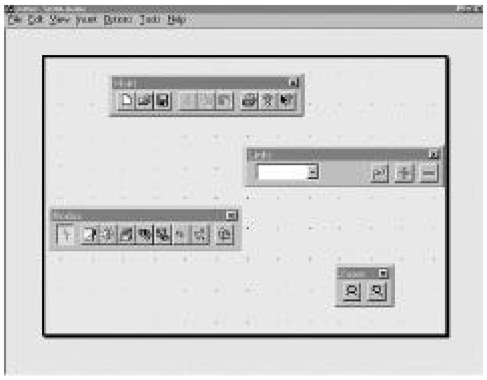
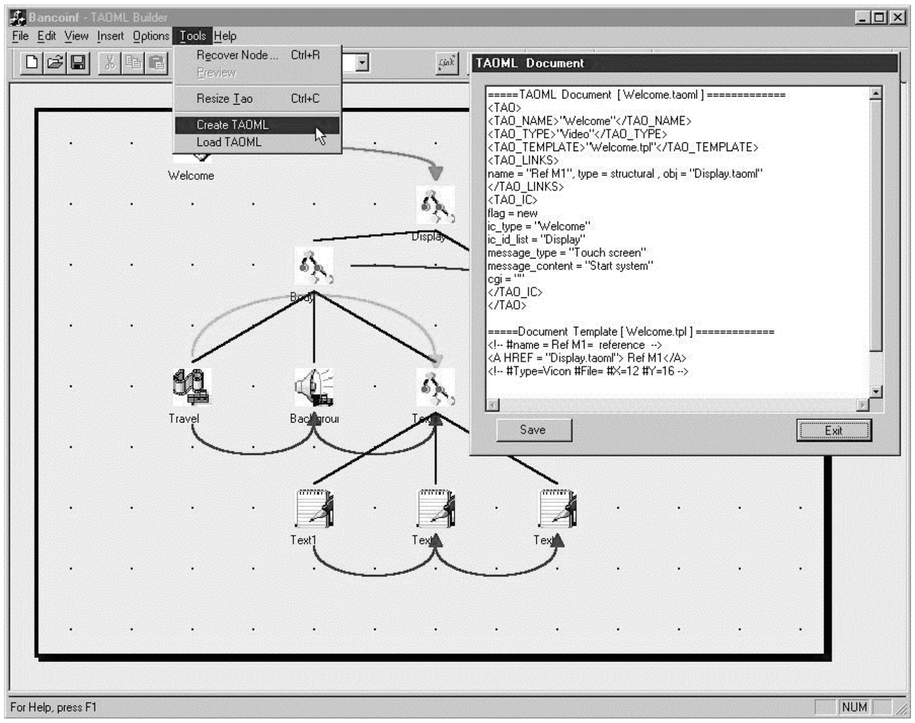


Figure 3. TAOML Builder Toolbars.



```
=====TAOML Document [ Welcome.taoml ] =============
<TAO>
<TAO_NAME>"Welcome"</TAO_NAME>
<TAO_TYPE>"Video"</TAO_TYPE>
<TAO_TEMPLATE>"Welcome.tpl"</TAO_TEMPLATE>
<TAO_LINKS>
name = "Ref M1", type = structural , obj = "Display.taoml"
</TAO_LINKS>
<TAO_IC>
flag = new
ic_type = "Welcome"
ic_id_list = "Display"
message_type = "Touch screen"
message_content = "Start system"
cgi = ""
</TAO_IC>
</TAO>

=====Document Template [ Welcome.tpl ] =============
<!-- #name = Ref M1= reference -->
<A HREF = "Display.taoml"> Ref M1</A>
<!-- #Type=Vicon #File= #X=12 #Y=16 -->
```

Figure 4. Figure 4 - Hypergraph and Matching TAOML.

The TAOML Builder has been tested on a selected sample of users that have shown the need to overcome some of the problems of using the graphical representation. The non-experienced users preferred to use the graphical representation of the hypergraph, experienced users preferred a textual representation. By selecting the command “Create TAOML” in the menu “Tools” TAOML Builder automatically generates the TAOML version of the hypergraph (Figure 4). This textual representation can then be edited from within the TAOML Builder. The user, therefore, can switch between textual and graphical editing of the hypergraph.

4. TAOML INTERPRETER

The TAOML Interpreter is a command line tool that interprets the TAOML output by the TAOML builder tool and generates valid HTML. The interpreter uses templates that are independent HTML pages to define the fundamental display element and location arrangement. For example, if the TAO is of image type, the template will just contain an HTML statement to introduce an image. If the TAO is of mixed type, the template will define some common parts and leave some space to insert the elements that are specific to this TAO.

The interpreter must also evaluate the link tag of TAOML. A link has attributes 'link_type', 'link_obj'. 'link_type' is either relational (spatial or temporal) or structural (COMPOSED OF). In the context of TAOML, a spatial link describes visible relationship between sub_objects inside one mixed object. For example, a mixed tao1 contains an image TAO2 and a text TAO3, then TAO1 has spatial link with both TAO2 and TAO3. A temporal link usually refers to an invisible object that is not a display element, but its activation time is influenced by the other. A structural link relates one TAO with another dynamically via user input or external input. For example, the user clicks a button in TAO1 will invoke another page TAO2, and then there is a structural link from TAO1 to TAO2.

For the associated index cell, the flag is "old" if the ic already exists, or "new" if the ic is to be created. The ic type, ic_id list, message type and message content can either be specified, or input by the user (indicated by a question mark in the input string). A corresponding HTML input form will be created so that the user can send the specified message to the ic's.

The TAO_HTML Interpreter can be presented in the following pseudo-code:

```
procedure interpreter(char *TAOname)
{
  open TAO definition file
  call TAO_parser() to construct the
     TAO data structure TAO_struct
  call template_parser(TAO_struct)
     to output HTML file
}
procedure TAO_parser(file_handle, link_type)
{
  while (not end of file)
  {
    read one line from the file
    distinguish tag and get information
       and store in data structure
  }
}
procedure template_parser(TAO_structure)
{
  if IC_PART is specified, output HTML statements
     to create a form to accept user's input and
     send message to the ic's through IC_Manager
  if template file exists
     open template file
  while (not end of file)
  {
    read one line from the file
    if (not <TAO_rel> tag)
      output html text
    else
    {
      get link_name from the <TAO_rel> tag
      search in the TAO_structure with link_name
      if (a link structure is found with the
      same link_name)
      {
 get link_type and link_TAO_name
 switch (link_type)
   case structural:
     insert <a href..> link in template
        to link with link_TAO_name
   case spatial:
     call procedure interpreter(link_TAO_name)
        to insert template of link_TAO_name
   }
    }
  }
}
```

5. IC BUILDER

The knowledge of a TAO-based mutimedia application is stored in a set of active index cells. The index cells can be created either before or after the TAOML has been created using the two TAOML tools. The IC Builder is a visual PC-based tool to help the user define active index cells. Once an index cell is defined, the IC Builder creates a formal specification file *.in (e.g. ic1.in). After all the index cells have been defined, the IC Builder generates a file ic.dat to characterize an application. This file ic.dat becomes the input to the next tool, the IC Compiler. The index cells specification files *.in, on the other hand, become the input to the customized IC Manager.

The main screen of the IC Builder is shown in Figure 6 of Chapter 8. As was said in Section 2, an IC may be seen as a kind of Finite State Machine. The IC Builder allows us to graphically specify the states and transitions of such a machine. Specifically, the IC Builder allows us to draw a state by clicking the corresponding icon in the tool bar, pointing the cursor at the desired position, and pressing the left button. The state will be numbered automatically. We may also delete a state or change the ID number of a state. We may draw or delete a transition between states and define a transition using the dialog shown in Figure 7 of Chapter 8. Clicking the Define_Transition icon in the tool bar, the user moves the cursor to the start position of one transition, then clicks the left button. A dialog appears on the screen. This dialog allows one to add as many transitions between two states as desired. Clicking the two buttons on the right side of the dialog allows the user to further define the input or output message of one transition between the two states. Figure 9 of Chapter 8 shows the dialog for defining an output message. We see that there are two columns in this dialog, the left one is used to define the action for the transition. Two fields are needed for each action, the action name (case insensitive) and the name of the file that contains the action. The right column defines the output messages in the transition. There are six options for the field "Output IC NO.".

"Specify an Existing IC ID". For this option, the user has to specify a positive integer as the IC ID.

"Send to a New IC". For this option, the corresponding output message will be post to an IC that will be activated when this message comes.

"Broadcast to All Ics". For this option, this message will be broadcast to all ICs. If the IC type in the field "IC type" is specified, the message will be broadcast to all ICs of the specified type. If not, the message will broadcast to all ICs that can receive the message.

"Contended by All Ics". For this option, this message will be contended by all ICs. If the IC type in the field "IC type" is specified, the message will be contended by all ICs of the specified type. If not, the message will be contended by all ICs which can receive the message.

"Broadcast to Selected Ics". For this option, this message will be broadcast to the selected ICs. The user has to program a function to compute the selected ICs. If the function needs to know the IC type, the user has to the IC type in the field "IC type".

"Contended by Selected Ics". For this option, this message will be contended by the selected ICs. The user has to supply a function to compute the selected ICs.

The IC diagrams definition need to be transformed to a so-called .in file when it is used as the input of the IC_Manager. Clicking the Export icon in the tool bar exports the diagram as a .in file. The export operation will create all the .in files in the project plus an ic.dat file for the input of the IC Compiler.

6. IC COMPILER

The IC Compiler is a command line tool that accepts an input file characterizing an application and generates the customized source code of the IC Manager. The default input file is ic.dat produced by the IC Builder tool. This file consists of a number of definitions with optional comments. Each definition type header is prefixed with a "$".

The supported definition types are the following. Header "$MESSAGE" defines input and output messages of an IC as: message_name/message_id. Header "$INCLUDE_FILE" allows the user to add include files to the application by giving the file name. Header "$ACTION/AUTO_GEN:YES|NO" defines actions of an IC as: action_name/action_id[/function_name[/file_name]]. If AUTO_GEN is YES, a source code file for actions is automatically generated by gathering the functions in the given files. Otherwise, the user must supply the file. Header "$IC_ID" defines IC_IDs as: name_of_ic_id/number. Header "$MUST_FUNC/ AUTO_GEN:YES|NO" defines functions that are necessary in IC Manager as: func_group_name[/file_name]. Once again AUTO_GEN equal to YES will automatically create the needed file from a given list of files containing functions. The functions are obtained by specializing system-provided templates. Header "$THRESHOLD" defines thresholds for fuzzy computation as: ic_type/fuzzy_number. Header "$DB_ACCESS" defines the access to a database as: view_name/database/tables /attributes/condition where

tables = table [,table]*
attributes = attribute [,attribute]*
condition is a predicate.

If database is "DEFAULT", it means to access the default database that is defined in the program. If attributes is "*", it means all attributes of the specified tables. If condition is "NULL", it means that the generated SQL has no condition. The definition will cause an SQL command for the specified database of the following type to be created:

SELECT attributes FROM tables WHERE condition

After execution of the IC Compiler, the active index structure of the MICE application is ready to be exercised by the IC Manager tool.

7. IC MANAGER

The IC Manager is a run-time tool that receives incoming messages, activates index cells, performs actions, and handles outgoing messages. Each message sent from one IC to another passes through the IC Manager. Another implementation would have each IC as a separate process, however this would result in high interprocess communication overhead. In order to avoid this overhead in the prototyped application, the MICE approach avoids these separate processes.

The IC Manager contains both domain-independent and domain-specific parts. The domain-specific part contains the user-defined procedures used to perform predefined actions. The domain-specific part also controls the external messages sent to the IC Manager. The separation between domain-independent and domain-specific parts makes implementation of a multimedia application containing powerful actives indexes easy since only the domain-specific parts need be given.

Since the IC Manager is written in standard C language, the MICE workbench can be used to develop applications intended for deployment on both PC and UNIX based web servers. The IC Manager has been used to prototype a Smart Image System, Web Browser Monitor [Chang96c], B-Tree, and Medical Personal Digital Assistant [Chang96b].

8. TAOML TO XML TRANSLATOR

TAOML can be described as a subset of XML [W3C98], the Extensible Markup Language. Like HTML, XML is based on the Standard Generalized Markup Language (SGML) [ISO92]. But while HTML is a non-extensible grammar, XML is designed to be extensible, while at the same time avoiding some of the complexity of SGML. Microsoft’s proposed Channel Definition Format for push technologies is an example of an XML application. Using XML rather than HTML would essentially allow us to avoid the TAOML interpreter. Also, the flexibility of XML links (including the possibility of embedding one document inside of another and bidirectional links) corresponds much more closely to the hypergraph model of TAO.

The major differences between HTML and XML are as follows:

- *Hierarchical element structure*: XML documents must have a strictly hierarchical tag structure. Start tags must have corresponding end tags. In XML vocabulary, a pair of start and end tags is called an element.

- *The empty tag requires trailing slash*: Empty tags are also allowed as elements in XML documents. An empty tag is essentially a start and end tag in one, and is identified by a trailing slash after the tag name.
- *Single root element*: XML documents allow only one root document. This restriction makes it easier to verify that the document is complete.
- *Quoted attribute values*: All attribute values must be within single or double quotes.
- *Case sensitivity*: XML tags are case-sensitive.
- *Relevant white space:* White space in the data between tags is relevant, because XML is a data format.
- *Extensibility*: XML can be extended by creating new tags that make sense.

The Advantages of using XML are:

- Authors and providers can design their own document types using XML, instead of being stuck with HTML.
- Information content can be richer and easier to use, because the hypertext linking abilities of XML are much greater than those of HTML.
- XML can provide more and better facilities for browser presentation and performance.
- XML removes many of the underlying complexities of SGML in favor of a more flexible model, so writing programs to handle XML will be much easier than doing the same for full SGML.
- Information will be more accessible and reusable, because the more flexible markup of XML can be used by any XML software instead of being restricted to specific manufacturers as has become the case of HTML.
- Valid XML files can be used outside the Web as well, in an SGML environment.

The TAOML-to-XML Translator works in the following way:

```
procedure TAOML-to-XML translator
begin
    open TAOML page
    while (not end of file) do
    begin
        read one line from the input file
        recognize tag
        convert into appropriate XML tag
        write into output file
    end
end
```

To implement the TAOML-to-XML Translator, we need a DTD (Document Type Declaration), which is a grammar that describes what tags and attributes are valid in an XML document, and in what context they are valid. DTD specifies which tags are allowed within certain other tags, and which tags and attributes are optional. With regard to a DTD, an XML document can: 1) refer to a DTD using a URI, or 2) include a DTD inline as part of the XML document, or 3) omit a DTD altogether.

The DTD for TAOML is as follows:

```
<!ELEMENT TAO (TAO_NAME, TAO_TYPE, TAO_TEMPLATE,
   (TAO_LINKS)*, (TAO_IC)?,  (TAO_SENSI)?,  (TAO_DATA)? )>
<!ELEMENT TAO_NAME (#PCDATA)>
<!ELEMENT TAO_TYPE (#PCDATA)>
<!ELEMENT TAO_TEMPLATE (#PCDATA)>
<!ELEMENT TAO_LINKS EMPTY>
<!ATTLIST TAO_LINKS
      name         CDATA
      type         CDATA
      obj          CDATA
>
<!ELEMENT TAO_IC EMPTY>
<!ATTLIST TAO_IC
      flag         (old|new)
      ic_type      CDATA
      ic_id_list   CDATA
      message_type   CDATA
      content      CDATA
      cgi          CDATA
>
<!ELEMENT TAO_SENSI  (#PCDATA)>
<!ELEMENT TAO_DATA   (#PCDATA)>
```

The TAOML-to-XML Translator is implemented in PERL. As an example, the following TAOML is the input:

```
<TAO>
<TAO_NAME> "activate1" </TAO_NAME>
<TAO_TYPE> mixed  </TAO_TYPE>
<TAO_TEMPLATE> "activate1.tpl" </TAO_TEMPLATE>
<TAO_IC>
 flag       = new
 ic_type     = "PR"
 ic_id_list   = ""
 message_type = "M0"
 content      = ""
 cgi          = "corba.cgi"
</TAO_IC>
</TAO>
```

The output XML is as follows:

```
 <?xml version = "1.0" ?>
<!DOCTYPE TAO SYSTEM "mse.dtd">
<TAO>
<TAO_NAME> "activate1" </TAO_NAME>
 <TAO_TYPE> mixed </TAO_TYPE>
 <TAO_TEMPLATE> "activate1.tpl" </TAO_TEMPLATE>
 <TAO_IC flag =" new" ic_type = "PR" ic_id_list = "" message_type = “M0"
content = "" cgi="corba.cgi" >
</TAO_IC>
</TAO>
```

9. DISCUSSION

A visual software engineering environment for multimedia applications has been developed in order to study the visual software development process. Prototype systems have been developed using the MICE tools. Preliminary studies have shown that while novice users especially appreciate the visual environment, expert users prefer to have the option to work visually as well as textually within the same tool. The MICE approach allows for powerful applications to be quickly prototyped.

In the future, the tools will be more closely integrated resulting in a seamless MICE developer’s environment. This environment should provide a closer linkage between the design of the ICs and the design of the TAOML as well as automate the transfer of the output of the TAOML Builder and IC Builder tools to the Web Server where the IC Compiler and TAOML Interpreter are hosted. In addition, since the TAOML Builder is based on an underlying Symbol Relation Grammar, a syntax-directed version of the tool will be built.