MULTIMEDIA SOFTWARE ENGINEERING

Shi-Kuo Chang

Springer Science+Business Media, LLC

MULTIMEDIA SOFTWARE ENGINEERING

THE KLUWER INTERNATIONAL SERIES IN SOFTWARE ENGINEERING

Series Editor

Victor R. Basili
University of Maryland
College Park, MD 20742

Also in the Series:

FORMAL SPECIFICATION TECHNIQUES FOR ENGINEERING MODULAR C PROGRAMS, *by TAN Yang Meng;* ISBN: 0-7923-9653-7

TOOLS AND ENVIRONMENTS FOR PARALLEL AND DISTRIBUTED SYSTEMS, *by Amr Zaky and Ted Lewis;* ISBN: 0-7923-9675-8

CONSTRAINT-BASED DESIGN RECOVERY FOR SOFTWARE REENGINEERING: Theory and Experiments, *by Steven G. Woods, Alexander E. Quilici and Qiang Yang;* ISBN: 0-7923-8067-3

SOFTWARE DEFECT MODELING, *by Kai-Yuan Cai;* ISBN: 0-7923-8259-5

NON-FUNCTIONAL REQUIREMENTS IN SOFTWARE ENGINEERING, *by Lawrence Chung, Brian A. Nixon, Eric Yu and John Mylopoulos;* ISBN: 0-7923-8666-3

EXPERIMENTATION IN SOFTWARE ENGINEERING: *An Introduction*, by Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén; ISBN: 0-7923-8682-5

The Kluwer International Series in Software Engineering addresses the following goals:

- To coherently and consistently present important research topics and their application(s).
- To present evolved concepts in one place as a coherent whole, updating early versions of the ideas and notations.
- To provide publications which will be used as the ultimate reference on the topic by experts in the area.

With the dynamic growth evident in this field and the need to communicate findings, this series provides a forum for information targeted toward Software Engineers.

MULTIMEDIA SOFTWARE ENGINEERING

by

Shi-Kuo Chang
University of Pittsburgh, U.S.A.

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Library of Congress Cataloging-in-Publication Data

Chang, S. K. (Shi Kuo), 1944-
Multimedia software engineering / by Shi-Kuo Chang.
p. cm. -- (Kluwer international series in software engineering ; 7)
Includes bibliographical references and index.
ISBN 978-1-4613-6997-4 ISBN 978-1-4615-4435-7 (eBook)
DOI 10.1007/978-1-4615-4435-7
1. Multimedia systems. 2. Software engineering. I. Title. II. Series.

QA76.575 C43 2000
006.7'6--dc21

99-056039

Preface

Multimedia has two fundamental characteristics that can be expressed by the following formula: *Multimedia = Multiple Media + Hypermedia.* How can software engineering take advantage of these two characteristics? Will these two characteristics pose problems in multimedia systems design? These are some of the issues to be explored in this book.

Managers, software engineers, programmers and people interested in gaining an overall understanding of multimedia software engineering can read the two to four chapters of the book.

The six chapters from Chapter Three to Chapter Eight present multimedia software engineering according to the conceptual framework introduced in Chapter One. Practitioners, system developers, multimedia application designers, programmers and people interested in prototyping multimedia applications can read these six chapters.

The next three chapters are more research oriented and are mainly intended for researchers working on the specification, modelling and analysis of distributed multimedia systems. Scientists, researchers and software engineers interested in the systems and theoretical aspect of multimedia software engineering can read these three chapters.

The book can be used as a textbook in a graduate course on multimedia software engineering or in an undergraduate course on software design where the emphasis is on multimedia applications. It is especially suitable for a project-oriented course. To serve that purpose, exercises and project suggestions are included, and the experimental MICE software can be downloaded from my web site at http://www.cs.pitt.edu/~chang.

For an evolving discipline such as multimedia software engineering, the courseware will also be constantly evolving so that a multitude of

information items can be accessible to the readers and viewers. I am also committed to be available on the Internet in case the readers and viewers have any questions regarding this book or the MICE experimental software.

I am indebted to my colleagues at the University of Pittsburgh who supported my idea to redesign the second graduate elective on software engineering into a course on multimedia software engineering. Thanks are also due to my students in the Spring 1999 multimedia software engineering class: Deepika Balakrishna, Kristin Balon, Glenn Buckholz, Jing Chen, Patrick Herron, Meepani Karunanayake, Xiaozhong Luo, Matthew McGrath, Thongchai Rojkangsadan, Xiaodong Shi and Harry Thompson who contributed to the survey reported in Chapter Two and also made many suggestions regarding the MICE experimental software. I often feel that I learn more from them than what they learn from me. Undoubtedly, that is why teaching is such a rewarding profession.

Regarding the contents of this book, individual chapters are based upon my own research and the research work of my colleagues or former students: Tim and Mara Arndt (Chapter Seven and Chapter Eleven), Diana Chang (Chapter Six), Chi-Cheng Lin (Chapter Ten) and Peppe Polese (Chapter Nine).

The editing of this book was accomplished on a tiny Toshiba Libretto, which apparently was not prepared for such twenty-hour-a-day heavy editing work during the hot summer months. It moaned and cried and managed to destroy two chapters when overheated, but eventually delivered the work. I am indebted to James Chien who restored one of the two destroyed chapters and saved me at least one week of hard work.

I also thank my wife Judy, my daughters Emily and Cybele and in particular my grandson Albessant who let me sneak away to complete this book during a critical period of Albessant's life, although I must admit it is much more fun to watch Albessant grow.

Shi-Kuo Chang

Chapter 1

A Framework for Multimedia Software Engineering

On December 31, 1998 USA Today carried an interesting article, "Birth of a New Order", talking about the year that the world's lines of time and space collapsed. The most incisive paragraphs are excerpted below:

The global, time-crunched market driven by electronic information "forces things to get bigger and smaller at the same time," says Nicholas Negroponte, author and technologist at the Massachusetts Institute of Technology. "And that's so ironic, when things want to do both but not stay in the middle. There will be an increasing absence of things that aren't either very local or very global". Oil and cars aren't much suited to being small and local. So they're moving to become gigantic and cross-border. As for being small and local, that's where the Internet, or World Wide Web, comes in -- and it works in two ways. It lets little companies be global, so a start-up in a garage can put its goods or services on a Web site and sell world-wide, competing against midsize or big companies, wiping out disadvantages (such as distribution and scope) that once had to do with distance. And since little companies can change direction faster than bigger ones, they have an advantage in time. Big companies used to have time and distance on their side. Increasingly, little ones do. And so in 1998, we had the phenomenon of Amazon.com, which has become such a symbol of small beating big that business people have turned it into a verb: to be "amazoned".

It is interesting to study how Internet and multimedia technology might help the "little guys" compete against the "big guys". Indeed, this investigation may lead to a better understanding of the roles of multimedia software engineering (MSE) in this new Internet-based industrial revolution.

1. HIGH PRESENCE AND HIGH TOUCH

Internet and multimedia are changing the rules of the economy and redefining our businesses and our lives. It is destroying solutions such as mass production, segmented pricing, and time and distance for big businesses. A company can develop a web page and advertising campaign and quickly compete in the world market. This has led to the flattening of the economy, whereby established companies and individuals doing business on their own can compete on an equal plane. The small companies that succeed in challenging the large companies are the ones who can maintain a global presence and yet make people feel that they are personal and easy to deal with.

(1) Small companies can interact closely with their customers, so that the customers feel that they are able to communicate to the small company what they need, as opposed to the customers merely accepting the mass-produced product that large companies will sell and not give much ground for derivation from the product.

(2) Web changed from just a means of advertising, to a medium to rapidly exchange ideas with potential customers. Since the small company listens to what they say, it not only results in having a satisfied (and probably a faithful) customer but increases sales significantly with time.

(3) The Internet's primary advantage in advertising is not so much in attracting attention and conveying a brief message (the tasks assigned to traditional advertising media), but lies instead in delivering in-depth, detailed information. Its real power is the ability to provide almost infinite layers of detail about a product or service, interactively, at the behest of the user.

However, small companies have to work smarter and respond more quickly [Murr98]. They have to avoid mistakes and make the best of possible use to everything. Corporations with big budgets can afford to lose their investments, while a small company looks at web as survival, not as an investment.

The small businesses also need to realize having a web site does not automatically mean that the company will reach millions of potential customers. It simply means that there is the potential to reach millions of potential customers. Company has to promote the site through advertisements, e-mail, links to other sites, and cutting edge multimedia technology to attract lots of visitors. For a new start-up small company, a brand new idea is always crucial. Second, multimedia technology should be used to provide various kinds of services on the web site. Third, once the site starts catching on and e-mails start rolling in, more and more person hours should be put into keep up with it all.

2. WHAT BUSINESSES WANT FROM MULTIMEDIA TECHNOLOGY

2.1 How Small Businesses View Technology

There are significant differences in how large and small businesses view technology [JBR95]: 1) Affordability - small businesses have to be extremely cost conscious, while big businesses have a larger capital to invest in technology. 2) Scalability - all small businesses have ambitions to become big and this is an important requirement in the technology that they buy. 3) Fast return on investment - while large companies can wait up to 12-18 months for returns, small businesses want instant gratification. 4) Simplicity - most small businesses want 'plug and play' products such as, for example, the Unix multiserver networks, or the peer-to-peer networks.

2.2 Advantages of Multimedia Technology

From the perspective of a small company, the advantages of multimedia technology are perceived as follows:

(1) Helps develop advertising that could be used in many different media, thus cutting advertising costs.

(2) Cuts down on the amount of time the development staff needs to deal with customer service issues.

(3) Gives the appearance of having all the customer service support of a larger company.

(4) Facilitates out-sourcing [Neil98].

(5) Keep clients aware of progress in almost real time by allowing them access to the site in development.

2.3 Wanted: Flexible MSE Tools

What businesses want from multimedia, in the above context, become quite clear:

(1) Affordability $\Rightarrow$ software tools
(2) Scalability $\Rightarrow$ scalable software tools
(3) Fast return on their investment $\Rightarrow$ prototyping tools
(4) Simplicity $\Rightarrow$ easy-to-use tools
(5) Helps develop advertising that could be used in many different media $\Rightarrow$ adaptive multimedia tools
(6) Cuts down on the amount of time the development staff needs to deal with customer service issues $\Rightarrow$ customer-service-oriented tools

(7) Gives the appearance of having all the customer service support of a larger company ⇒ scalable tools
(8) Facilitates out-sourcing ⇒ specification tools
(9) Keep clients aware of progress in almost real time by allowing them access to the site in development ⇒ incremental development tools

3. INTERNET AND MULTIMEDIA TECHNOLOGY TRENDS

To support the design of such flexible MSE tools, we note the following Internet and multimedia technology trends:

(1) The Browser will become the preferred universal interface.

(2) Java, already known as the de facto standard, will offer more attractive features in reducing the cost of Internet development to that of typical client/server projects.

(3) Event-based modeling will provide software developers significant advantages over in-house development.

(4) Sophisticated tools will monitor events as they change. Agents will be used to post events and make their own decisions about how to process events [Blak98]. Window dialogs will assist engineers with dynamics of objects.

(5) Businesses will find they will not be equipped to keep-up with new technology. To compensate, they will defer to out-sourcing to obtain Internet and multimedia technology.

(6) The proliferation of the Internet will give rise to data centers that will decentralize data and provide multiple companies access to data on a global basis [Neil98]. This in turn will promote the development of higher speed communication lines with remote management software systems. Internet 2 will replace existing multimedia standards with higher speed and enhanced video-conferencing.

(7) Embedded wireless communication will find its way into the Internet. This technology will facilitate remote access to the Internet, rendering further proliferation of its use [Patr99].

(8) Personal digital assistant (PDA) and/or palm top computer will become popular because of their cheap price and small size. People will use PDA to connect to Internet or do personal information processing. It can display video clip, play audio file or control household equipment. Mobile agent software will be the important application for the PDA. Such agent software can do various activities such as downloading or finding interesting information from Internet, exchanging information among them, etc.

(9) There will be multimedia components in the software engineering process.

(10) Multimedia software in the future will be multi-lingual in order to gain widespread usage rather than specific in any particular language.

4. WEB SITE LIFE CYCLE

A web site in many ways resembles other types of corporate information systems. Each web site has a limited life span, similar to the water fall software life cycle model. One major difference is the emphasis on content development in multimedia applications. The phases of web site development are as follows: idea formulation, general web site design, detailed design of web site, testing of an implementation and maintenance.

(1) Idea Formation: During the idea formulation phase, specific target-marketing program, content goals and objectives must be set. Since a web site development project can become very time consuming and a major capital investment to owners of small businesses, it may be more effective to identify opportunity of specialized markets big companies have ignored. Furthermore, the profile of netizens must be carefully studied [Choi99] to find out who is surfing the net and what these people are looking at. Small businesses should be aware of the dynamics of the on-line market place and develop strategies and plan accordingly. The ideas of this phase can pave the foundation for developing a comprehensive plan for web site design.

(2) Web Site Design: Web site should be integrated into the company's backbone information system so that the web site can grow along with the business. To be successful, companies must integrate e-commerce into their overall business strategies and processes. Moreover, content needs to be targeted to specific user's needs. Visitor's information should be collected so that the company will be able to tailor the web pages to the specific needs of the interested customers. Furthermore, it is important that the web site can be surfed fast and efficiently. In addition, the users should be involved by providing an opportunity for them to input suggestions and complaints. The development of navigational cues and the user interface is of critical importance. The actual design tasks can be out-sourced for a small company. Also a new web site should be linked to as many search engines as possible. This can increase the chance that the web site is visited. Financial infrastructure should be developed properly as well.

(3) Testing: Once the implementation is complete, the company should conduct a pilot to test its integrity and effectiveness. The pilot provides an opportunity to obtain feedback from functional groups, customers and business partners. It ensures the quality and usability of the site.

(4) Maintenance: It is essential that new content is developed and the web site is kept refreshed. Timeliness is the key on the web. Moreover, appointing a web master to manage the site on a day-to-day basis is imperative. Web master can trouble-shoot any error such as a link to a defunct web address, track the traffic of the web site, use reader feed back to build a loyal following and ensure server maintenance and security. Also, this person or persons should make sure that the company's web site supports the latest versions of popular browsers.

5. DUAL ROLES OF MULTIMEDIA SOFTWARE ENGINEERING

Having discussed what businesses want from multimedia technology and the web site life cycle, we can now discuss the roles of multimedia software engineering. We can view MSE in two different, yet complementary, roles: 1) to apply multimedia technology to the practice of software engineering; and 2) to apply software engineering principles to the design of multimedia systems.

Multimedia has two fundamental characteristics that can be expressed by the following formula: *Multimedia = Multiple Media + Hypermedia.*

How can software engineering take advantage of these two characteristics? Will these two characteristics pose problems in multimedia systems design?

In Chapter 2 we will give a focussed survey of current research in MSE to apply multimedia technology to the practice of software engineering, or to apply software engineering principles to the design of multimedia systems. From the focussed survey of Chapter 2 it will be seen that multimedia is useful in software engineering, but whole-hearted incorporation of multimedia in software engineering has not yet happened [Hira99]. There is an ongoing paradigm shift -- from business orientation to entertainment orientation [Hira99]. New software process models and paradigms, such as object-oriented approach, are needed in multimedia systems design. Other interesting approaches include model-based approach to define navigation and access primitives, virtual multimedia objects approach to construct complex multimedia objects using virtual links, and identification of patterns (of navigation, news, landmark, etc.) to facilitate multimedia design. A long-term goal of MSE should be to design multimedia systems by multimedia.

6. A CONCEPTUAL FRAMEWORK FOR MSE

A conceptual framework for MSE based upon the notion of *multidimensional language* (ML) will now be presented.

A multidimensional language is a language where the primitives are objects of different media types and the operators comprise of spatial and temporal operators. Because of the importance of such spatial/temporal operators, we prefer to call such languages multidimensional languages rather than multimedia languages, although the multidimensional languages can be used to specify multimedia applications. From this viewpoint, a multimedia application is equivalent to a multidimensional language ML.

This viewpoint enables us to describe the various aspects of multimedia applications with conceptual clarity. The corresponding framework for MSE thus provides a principled approach, a set of scalable, adaptive tools and an environment for the specification, design, testing and maintenance of multimedia applications.

6.1 Syntactic Aspect

A multimedia application is constructed from a collection of multimedia objects. The primitive objects are media objects of the same media type. The complex multimedia objects are composed from these primitive objects and in general are of mixed media types. The syntax of ML describes how the complex multimedia objects are constructed from the other multimedia objects. Spatial and temporal composition rules must be taken into consideration.

6.2 Semantic Aspect

Multimedia applications nowadays are seldom passive. A passive multimedia application can be specified by a static ML, but a dynamic multimedia application requires the system to take actions in response to user input or internal/external stimuli. The semantics of ML describes how the dynamic multimedia objects are derived from other multimedia objects when certain internal/external events occur. Since an important characteristics of multimedia is the ability to create links and associations, the semantics of ML must take that into consideration.

6.3 Pragmatic Aspect

Multimedia applications are heavily content-based and require a lot of manual hard work to put together. Tools are needed to assist the designer in

building a multimedia application in a timely fashion. The pragmatics of ML can be based upon the patterns for various multimedia structures or sub-structures, such as navigation structures, content-based retrieval structures, etc. Once such structures and sub-structures are identified, they can be used as building blocks in putting together a multimedia application.

6.4 Systems Aspect

Last but not least, the systems aspects of multimedia applications must be considered. Multimedia applications require the support of distributed multimedia systems. The systematic design, specification, analysis and optimization of distributed multimedia systems will improve the performance of multimedia applications. Both QoS (quality of service) and QoP (quality of presentation) must be considered in systems design.

6.5 The Multimedia Software Life Cycle

With the above described framework, the multimedia software life cycle can be seen to consist of three phases: 1) **Syntactic Phase**: Gather user's requirements to specify the syntactic structure of the multimedia application. 2) **Semantic Phase**: Design the actions to be performed by the multimedia application. 3) **Pragmatic Phase**: Identify the basic building blocks and utilize tools to implement and test the application.

In any one of these three phases, the designer must always pay attention to the systems issues and to optimize the performance of the application in a distributed multimedia system environment.

If the three phases are followed in a sequential order with no repetition, we have something similar to the classical waterfall model. If the three phases are iterated fairly quickly in a software development environment with a set of integrated tools, we have something akin to the rapid prototyping model. If the three phases are iterated using increasingly sophisticated tools with more and more emphasis on scaled-up operations, we have the spiral model.

7. ORGANIZATION OF THE BOOK

This book is organized according to the conceptual framework described in Section 6.

7.1 Overview

Chapters 1 and 2 give an overview of multimedia software engineering. These two chapters are introductory in nature and can be read with ordinary effort. Managers, software engineers, programmers and people interested in gaining an overall understanding of multimedia software engineering can read mainly these two chapters and secondarily the following two chapters, chapters 3 and 4, on the syntactic aspect of multimedia software engineering.

7.2 Syntax, Semantics and Pragmatics

The next six chapters, chapters 3 to 8, present the multimedia software engineering according to the proposed conceptual framework. Practitioners, system developers, multimedia application designers, programmers and people interested in prototyping multimedia applications can read these six chapters.

The syntactic aspect of multimedia software engineering is presented in chapters 3 and 4. Chapter 3 discusses the elements of a visual language, and Chapter 4 shows how the fundamental concept of a visual language can be extended to multimedia, so that multimedia interfaces can be designed. This chapter also gives a preliminary introduction to the concept of active index.

The semantic aspect of multimedia software engineering is the focus of the next two chapters. In Chapter 5 the active index is introduced. Chapter 6 then introduces the tele-action object, which combines the syntactic aspect (the multidimensional language for user interface), with the semantic aspect (the active index cells for performing actions).

Chapters 7 and 8 cover the pragmatic aspects of multimedia software engineering. The software tools useful in multimedia applications development are presented in Chapter 7. Chapter 8 describes the MICE (Multimedia Information Custom Engineering) environment in sufficient detail so that multimedia applications can actually be prototyped in the MICE environment. Such details can be skipped if the reader is not interested in using MICE to do actual prototyping work.

7.3 Research Issues

The next three chapters are more research oriented and intended for researchers working on the specification, modelling and analysis of distributed multimedia systems. Scientists, researchers and software engineers interested in the systems and theoretical aspect of multimedia software engineering can read these three chapters.

Chapter 9 discusses how multimedia languages can be designed systematically by adopting a principled approach. The design and analysis of distributed multimedia systems following a transformational approach is the subject matter of Chapter 10. Last but not least, Chapter 11 addresses the issues of formal specification of multimedia applications.

7.4 Exercises and Project Suggestions

The book can be used as a textbook in a course on software design or in a course on software engineering where the emphasis is on multimedia applications. To serve that purpose, exercises and project suggestions are included in Chapter 12. For exercises, projects and prototyping multimedia applications, the experimental MICE software can be downloaded from the author's web site at http://www.cs.pitt.edu/~chang.

8. COURSEWARE SUPPORT

To download the MICE experimental software, the reader can visit to the author's web site at: www.cs.pitt.edu/~chang, and follow the links to the courseware on multimedia software engineering. It can be seen that not only the MICE experimental software is available, but also a wealth of information on multimedia software engineering. For an evolving discipline such as multimedia software engineering, the courseware will also be constantly evolving so that a multitude of information items can be accessible to the viewers, readers and students. The author is also available on the Internet in case the reader has any questions regarding this book or the MICE experimental software. Please feel free to send e-mail to: chang@cs.pitt.edu.

Chapter 2

Perspectives in Multimedia Software Engineering

As discussed in Chapter 1, we can view multimedia software engineering in two different, yet complementary, roles: 1) to apply multimedia technology to the practice of software engineering, or 2) to apply software engineering principles to the design of multimedia systems.

Multimedia has two fundamental characteristics that can be expressed by the following formula: *Multimedia = Multiple Media + Hypermedia.*

Advantages of multiple media are: 1) full utilization of all senses (eye, ear. etc.), 2) dynamic presentations, and 3) better understanding by the user. The disadvantages include: 1) greater demands on storage, bandwidth and computing resources, 2) cognitive overload, and 3) system complexity.

Hypermedia is a style of building systems for information representation and management around a network of multimedia nodes connected together by typed links [Hala95]. The advantages of hypermedia include: 1) ease of documentation, 2) ease of conceptualization and/or visualization, and 3) dynamic expansion of information hyperspace. The disadvantages are also well recognized: 1) disorientation due to "lost in hyperspace" phenomenon, 2) cognitive overload, and 3) system complexity.

In this chapter we will give a focussed survey of current research in MSE to apply multimedia technology to the practice of software engineering, or to apply software engineering principles to the design of multimedia systems. Since multimedia is basically multimedia objects plus links, in the following survey the concept of *links and association* will come up time and again as the central theme. The survey is by no means exhaustive, but the topics presented in this survey are a fair representation of the current research issues in multimedia software engineering.

1. PROJECT MANAGEMENT USING MULTIMEDIA TOOLS

One successful application of multimedia technology to software engineering is in project management using hypermedia CASE tools [Wild98]. Since the traditional project management tools lack the ability to capture a multitude of decisions and do not provide document control, a new Decision Based Systems Development paradigm (DBSD) was developed [Wild91]. In decision based systems development, usually the following steps are to be taken:

- Identification and Articulation of the Problem
- Identify Alternative Solutions
- Choose Decision Criteria
- Justify Alternatives
- Evaluate Conditional Decisions
- Put the Decision into Context
- Build Decision View

Multimedia technology allows the decision makers to use text for problem description, graphs and diagrams r for representing problem space, and different colors and symbols for denoting status and the latest information. Moreover, hypermedia allows the linking of documents and people in a variety of ways.

In the Decision-based Hyper-multimedia CASE (DHC) tool, objects in an extended document base are linked by five types of links: 1) *Reference to problems/decisions in problem space* or SEE links, 2) *Reference to a single on-line document* or REFER on-line links, 3) *Reference to a Decision View of a document set* or VIEW links, 4) *Timed reference to a contact person* or CONTACT links, and 5) *References to off-line documents* or REFER off-line links.

Some link types may be motivated by unix/DOS system commands. For example REFER is similar to xloadimage for bit-mapped pictures in DOCS.

This DHC tool was applied to Low-Visibility Landing and Surface Operations (LVLASO) project at NASA Langley Research Center. The impact on LVLASO is that the DHC tool is applicable to early stages in systems development. Also, brainstorming was easily documented and not lost. It encourages people to be more goal-oriented so that tasks that did not clearly fit in were immediately dropped. Finally it keeps people up to date with decisions. Future additions may include a CONTACT hyperlink, group Decisions by functional areas, and better GUI. The importance of this project

is that it indicates the desirability of having many different types of links for information/people association.

2. SOFTWARE DOCUMENTATION

Another successful area of application of multimedia technology is in software documentation. By exploiting the nature of hypermedia, powerful multimedia-based program documentation systems can be developed.

2.1 RST Documentation Model

Reliable Software Technologies Documentation Model uses standard C comments with the addition of design documents on the WWW. Standard C Comments are added to program lines whose purpose is not clear, to the beginning of each function or procedure to explain its purpose and the Pre and Post conditions, and to the header function to describe its purpose.

The following information is also maintained:

- CVS revision history
- Requirements document
- Relevant research
- Architecture diagram
- UML diagrams

RST documentation web page includes the following information:

- Welcome to the new program web page
- Short description of the project goals and the specific problem to solve.
- Related links
- Project design document
- Project specifications document
- Architecture document
- Research Links: Link1, Link2, Link3

2.2 Linux HQ Kernel Documentation

Linux HQ Kernel Documentation provides the following:

- Hypertext transformation of the code
- Links to function definitions

- Function search engine

Similar to CVS file hierarchy with links to associated files, Linux documentation maintains alphabetic listing of file names. It supports a function search engine. Function calls are linked to function definition, and there are links to included header files. However, it still relies on the reader to understand the comments.

An example of Linux HQ Documentation of the Linux Kernel is illustrated in Figure 1. More information can be found at: http://www.linuxhq.com

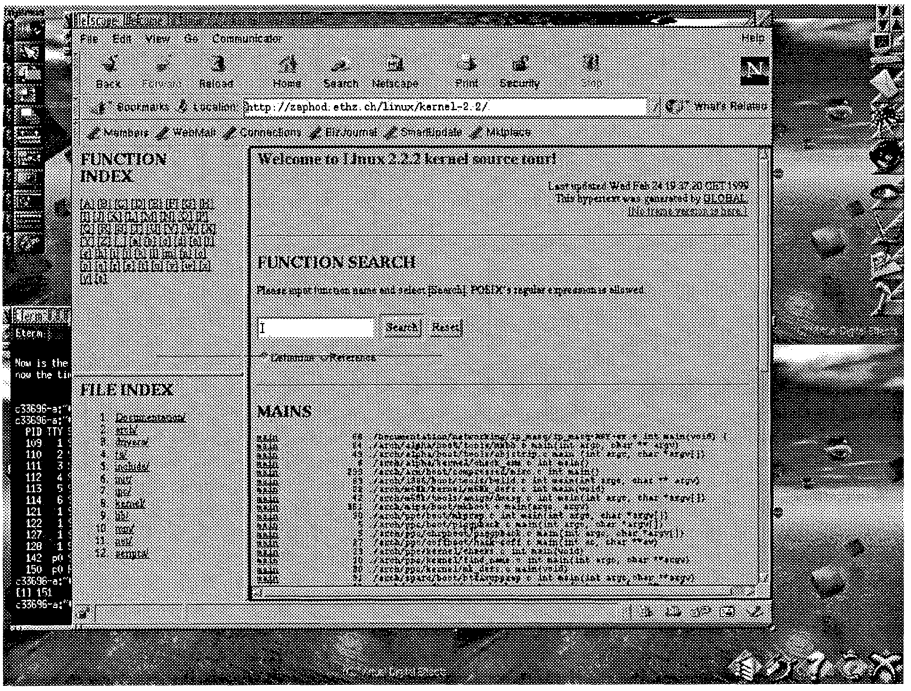

Figure 1. Documetation of the Linux Kernel – An example.

2.3 Variorum

The American Heritage Dictionary (1998) defines "variorum" as follows: "contains notes or comments by many scholars or critics". The creators of Variorum define it as a "multimedia tool that aids in the documentation of programs. ... The integration of WWW capabilities is a key aspect of variorum's usefulness".

Variorum [Chiu98] allows programmers to record the process of "walking through" codes using multimedia technology. Variorum supports hypertext transformation of code and the addition of programmer/author walkthroughs as voice annotations. Variorum modifies the source code to include annotation links. However its effectiveness depends critically on individual authors' annotation style. The amount of voice storage can also become excessive. However, in the future audio/visual software documentation systems may overcome the deficiencies of today's Variorum.

3. DESIGN OF MULTIMEDIA APPLICATIONS USING OBJECT-ORIENTED TOOLS

From the above survey it can be concluded that multimedia is useful in software documentation, but whole-hearted incorporation of multimedia in software engineering has not yet happened [Hira99]. There is an ongoing paradigm shift -- from business orientation to entertainment orientation [Hira99]. New software process models and paradigms, in particular the object-oriented approach, are needed in multimedia systems design. This section surveys several object-oriented approaches in multimedia systems design.

In what follows we will discuss:

- DAMSEL-Dynamic Multimedia Specification Language
- MET++-Multimedia Application Framework
- MME-Object-Oriented Multimedia Toolkit
- PREMO-Presentation Environment for Multimedia Standard

3.1 DAMSEL

DAMSEL is developed at the University of Minnesota. It includes the design and implementation of advanced multimedia constructs such as object-oriented extensible and temporal data model. It supports an execution environment based upon JAVA/CORBA. The temporal model describes along 3 axes: the temporal relations, delays and exec-based behavior. The representation using OO supports complex object definition and queries.

The DAMSEL execution environment is illustrated in Figure 2. The DAMSEL implementation architecture is illustrated in Figure 3.

The temporal model of DAMSEL is very flexible. It supports user, system, application-generated events and therefore enables very interactive dynamic application creation. In the execution architecture, specifications are written to define the environment behavior and run time execution is

determined by various events. DAMSEL specifications can be embedded with C++ or JAVA.

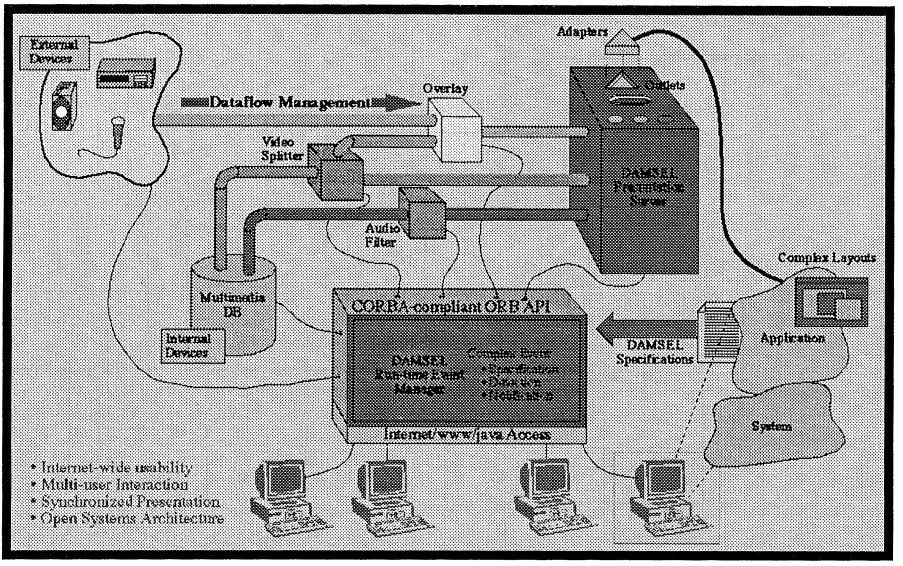

Figure 2. DAMSEL execution environment.

Data flow model assists in analyzing and modifying multimedia data. Data is modelled as stream flowing from a media source to a sink. It allows the insertion of additional stream objects as the data flows. It also enables modification and analysis of data. The presentation model provides higher level abstraction for dealing with and controlling presentations. Data flow is connected to a presentation server that interacts with the application.

The heart of DAMSEL is a distributed client-server run-time event manager, which is a multi-user, interactive, internet-wide execution environment implemented using CORBA and JAVA.

More information about DAMSEL can be found at its web site:
http://www-users.cs.umn.edu/~pazandak/damsel.html

3.2 MET++

MET++ is an object-oriented application framework developed at the University of Zurich. It supports the development of multimedia applications using reusable objects for 2D, 3D graphics, audio, video, etc. The framework consists of a set of interconnected objects that provide the basic functionality of a working application, which can be easily specialized into individual application. Through subclassing and inheritance, it supports the reuse of

code as a class library and the reuse of design structures. Similar dependencies between object are pre-implemented through predefined object composition, event dispatching and message flow.

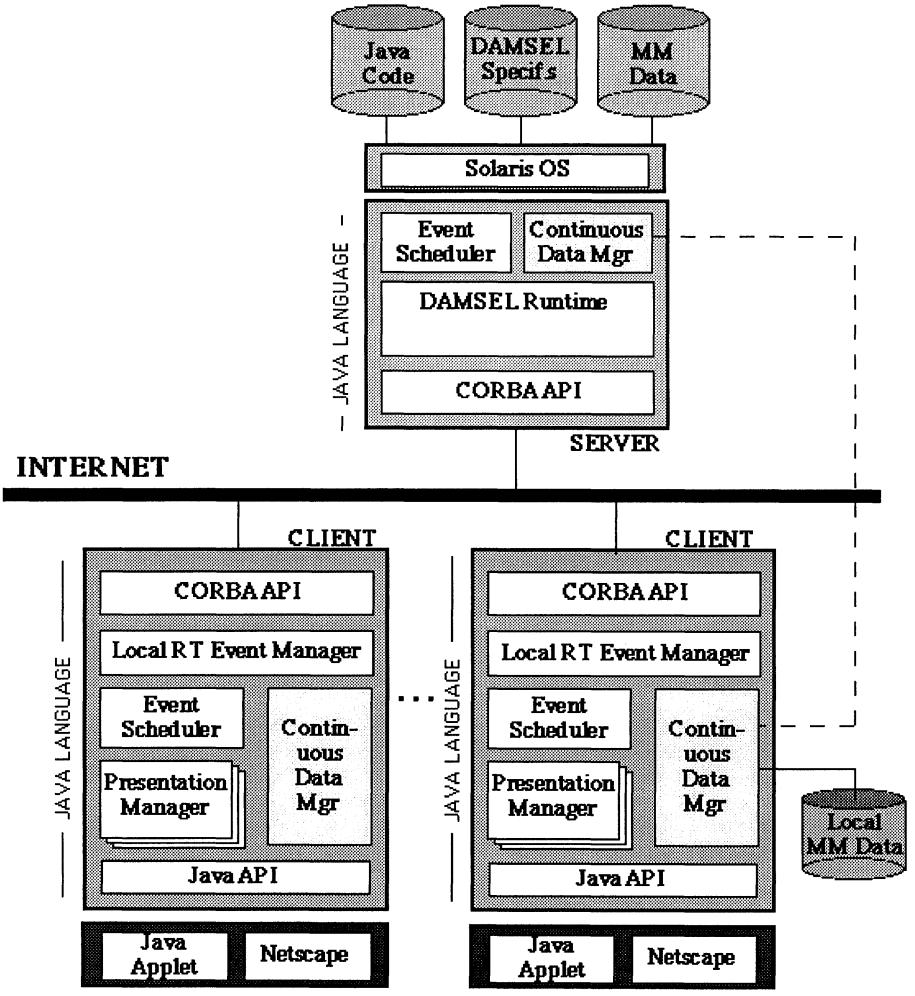

Figure 3. DAMSEL implementation architecture.

Figure 4 illustrates the MET++ architecture, which is based on OO application framework ET++. The ET++ OO class library integrates user interface building blocks, basic data structures, support for object I/O, and high-level application components. 3D graphics, audio classes, time structures are added for multimedia support. Hardware dependencies are hidden in a portability layer.

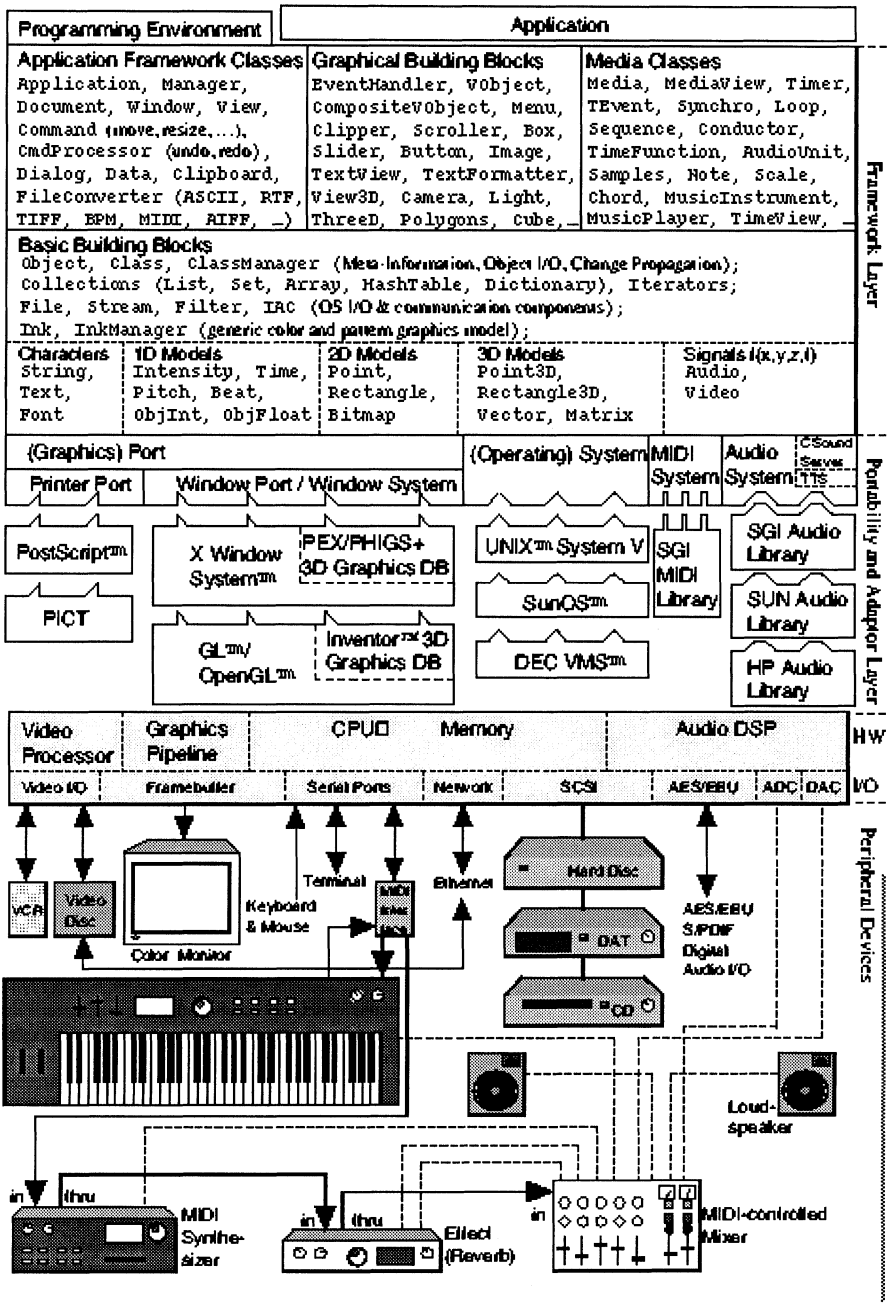

Figure 4. MET++ System Architecture.

Time synchronization is an important part of a multimedia presentation. In MET++, this is specified in a hierarchical composition. As illustrated by Figure 5, implicit information is made visually explicit due to the hierarchy.

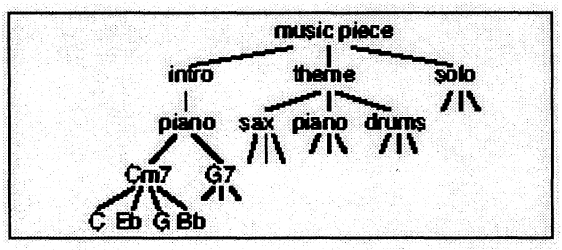

Figure 5. Temporal composition.

Example applications done using MET++ and more information about MET++ can be found at its web site:
http://www.ifi.unizh.ch/groups/mml/projects/met++

3.3 MME

The MME Multimedia Extension is a project by Computer Graphics Center (Fraunhofer), Germany. The software package features a class hierarchy and development tools for composing multimedia applications. The goals of MME are:

- OO modelling of various media
- encapsulation of distributed media access and control
- modelling of time as a precondition to define arbitrary temporal relations
- relations between media objects designed at a higher abstraction level and its realization at run time

MME objects have the following characteristics:

- application (AO), multimedia (MO)
- media objects handle the transfer of media data from a set of ports (source) to another
- set of ports (sinks)
- ports (devices like VCR, windows, files or sockets)
- complex media objects handle the definition and maintenance of relations (temporal and spatial) objects

MME is realized by executing:

- instantiation of media objects out of predefined multimedia classes
- instantiation of complex media objects to define spatial/temporal relations
- definition of new media classes as subclasses of predefined objects and classes

User interaction such as starting, stopping and cueing are supported in real time (interactive multimedia). MME is implemented in C++ on top of UNIX, Xwindows system with Xvideo extension, with about 4500 lines of code. The OO benefits include reuse, encapsulation, and less time to develop (about one man-year). More information about MME can be found at its web site:

http://zgdv.igd.fhg.de/www/zgdv-uig/software/MME

3.4 PREMO

PREMO (Presentation Environment for Multimedia Objects) is a new standard under development by ISO. The goals of PREMO are:

- to provide a general framework and a reference model for the creation and programming of distributed multimedia applications
- to allow existing media devices to be interfaced to an application
- OO programming infrastructure to support the development
- to recognize the evolution of multimedia system technologies for research tools to mature
- to certify products that meets QoS and fundamental requirements

PREMO is being developed at the CWI-Computer Center, Netherlands, and more information about PREMO can be found at its web site:

http://dbs.cwi.nl/cwwwi/owa/

4. SPECIFICATION OF MULTIMEDIA SOFTWARE SYSTEMS

Specification and design of multimedia applications pose new challenge to authoring systems due to temporal and spatial relations. Common design of hierarchical composition of objects needs to be found, thus leading to object-oriented tools.

For the specification of multimedia software systems a new paradigm is espoused: software engineers will do evolutionary design of complex

systems through: 1) architecture specification, 2) design rationale capture, 3) architecture V&V, and 4) architecture transformation, using an object-oriented architecture description language [Tsai99].

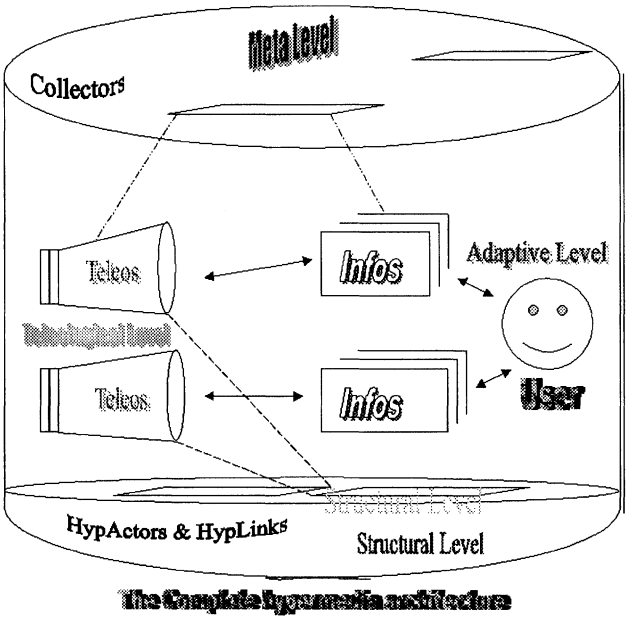

Figure 6. Hypermedia information system based upon the actor model.

Dattolo [Datto97] applies the actor model for modelling software as collections of distributed, cooperative entities, as illustrated in Figure 6. It is felt that the classical notion of object is too vague to support large-scale concurrency. On the other hand, actors combine object-oriented and functional programming in order to make the management of concurrency easier for the user. An actor reacts to the external environment by executing its procedure skills (scripts). An example for TeleoActor class definition based upon the ESAL (Extended Simple Actor Language) is as follows:

```
(Def TeleoActor
{Actor}
(stor info
hypServices image
inSuggestion brSuggestion cnSuggestion)
[(apply-filter), (visualize), (tree-brws), (grph-brws),])
```

The Dexter model for hypermedia is used, which is essentially a two-layer model – runtime layer and storage layer – for hypermedia. The architecture was applied in the development of a hypermedia system named DiBlue, which is a distributed version of Blue, a traditional OPLA hypermedia programming environment. It supports an object-oriented logic programming system in OPLA, a hybrid language originated from the marriage between Prolog and CLOS.

5. MODEL- AND PATTERN-BASED DESIGN APPROACHES

5.1 Model-based Approach to Hypermedia Design

In the model-based approach to hypermedia design, the key concept is to provide a comprehensive model for software specification and design. For instance, the Relationship Management Methodology (RMM) comprises 1) Entity-Relationship design, 2) Application diagram design, 3) M-slice (aggregate) design, 4) Navigational design, 5) User interface design, 6) Protocol conversion design, 7) Run-time behavior, and 8) Construction and testing [Isak96].

The Relationship Management Data Model (RMDM) is the cornerstone of the RMM methodology. It includes elements for representing information domain concepts such as entities and relationships, along with navigation elements such as links. As illustrated by Figure 7, an application design is described via an RMDM diagram.

During the E-R Design a study of the relevant entities and relationships of the application domain is conducted. These elements form the basis of the hypermedia application and show up as nodes or links. During the navigational design, relevant relationships are identified and made available for navigation. Since information units can have a very large number of attributes, the RMM groups the attributes into slices called M-Slices.

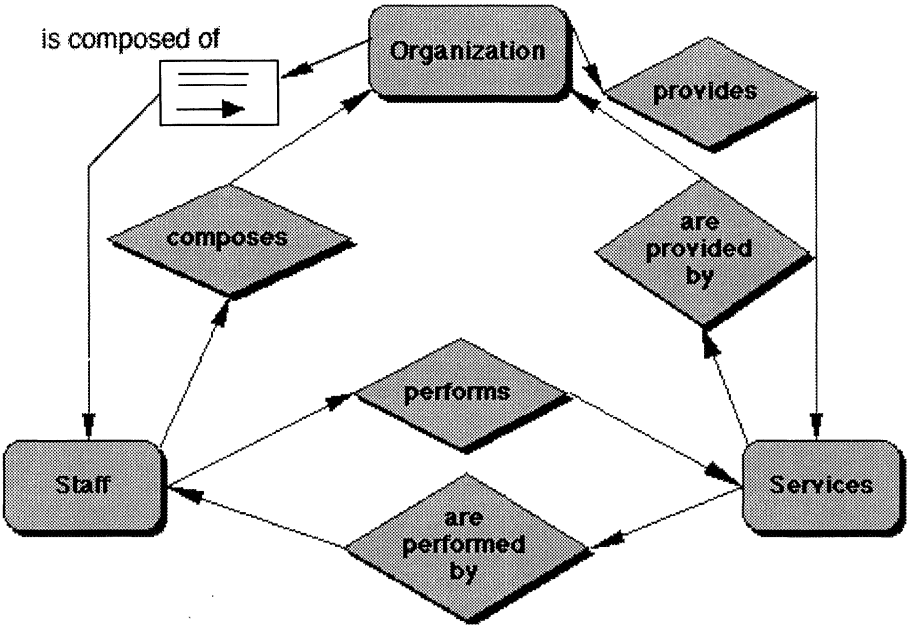

Figure 7. An RMDM diagram.

The way navigation is applied in the application by RMM is through the Access Primitives. The RMM Access Primitives include: 1) Unidirectional link, 2) Bi-directional link, 3) Grouping, 4) Conditional index, 5) Conditional guided tour, and 6) Conditional indexed guided tour.

Regarding links, navigation establishes an association between an original data component and destination data components. The associative link, or simply link, is what effectively allows navigation through data components, and represents the main characteristic of hypertext and hypermedia. The consideration of link types by definition establishes semantic information that allows hypermedia systems to efficiently manage data or data modelling, preventing this task to be the sole responsibility of designers or users. Link types can be used through almost every step of model-based approaches to hypermedia design.

What presently are found in most of the hypermedia systems are static, persistent and explicit links, defined over content. In RMM, there are links that can initiate a process such as email, a video exhibition, an audio exhibition, or a file download. Finally, *destination influence* establishes what of the operational services is going to be supported by the link. Links types are further broken down into two categories:

- Historical category (related to performed navigation)
- Speculative category (related to possible navigation)

A category that supports the retrieval service allows the presentation of the meaning associated to the description when it exists. *Destination influence* can provoke the definition of new links that were not initially related to infological modelling.

In conclusion, a framework is presented to categorize link types, which allows the design and construction of richer hypermedia applications. Link design, which is the kernel of model-based approaches to hypermedia design, is one of the areas that can benefit from the use of link types. Both information and operational elements influence link creation, so these issues should be considered during link design efforts. Information influence is related to conceptual relationships from the domain of interest, while operational influence is related to data manipulation services. It is also proposed that service and destination influences should be considered during link design. Links types and categories can also be identified and considered both with model based approaches and with authoring tools to hypermedia design.

5.2 Pattern-based Approach to Design

In the pattern-based approach to design the idea is to identify important and relevant design patterns, so that complex design can be constructed from these design patterns [Lyar97]. This approach can be applied to hypermedia applications [Garr97]. The patterns for hypermedia applications mainly deal with navigational structure and interface organization. The navigational contexts may include: class-derived context, link-derived context, composite-derived context, and arbitrary context. Two important navigational structures are *News* for new information and *Landmarks* for subsystems. In what follows, one such pattern will be discussed.

News is motivated by the following problem: "Given a large and dynamic web site, how do you tell users that there is new information or updates somewhere inside the site?" In huge web sites, it is not a good solution to compromise structure in order to make updates easy available. The solution is to structure the home page in such a way that a space is devoted to the newest additions:

- Presenting descriptive "headlines" regarding them
- Using those headlines as anchors to link them with their related pages

In summary a software engineering strategy for developing hypermedia applications is proposed, emphasizing the use of an object-oriented design method, the use of design patterns, and the partitioning of the development life cycle in a set of activities addressing different design concerns. The

design patterns may help to capture and reuse experience and to provide more comfortable and understandable navigation space.

5.3 Virtual Multimedia Object

Links can be explicit or virtual. For example, queries can be considered as virtual links. Multimedia objects can also be real or virtual. An example of virtual multimedia objects is illustrated in Figure 8.

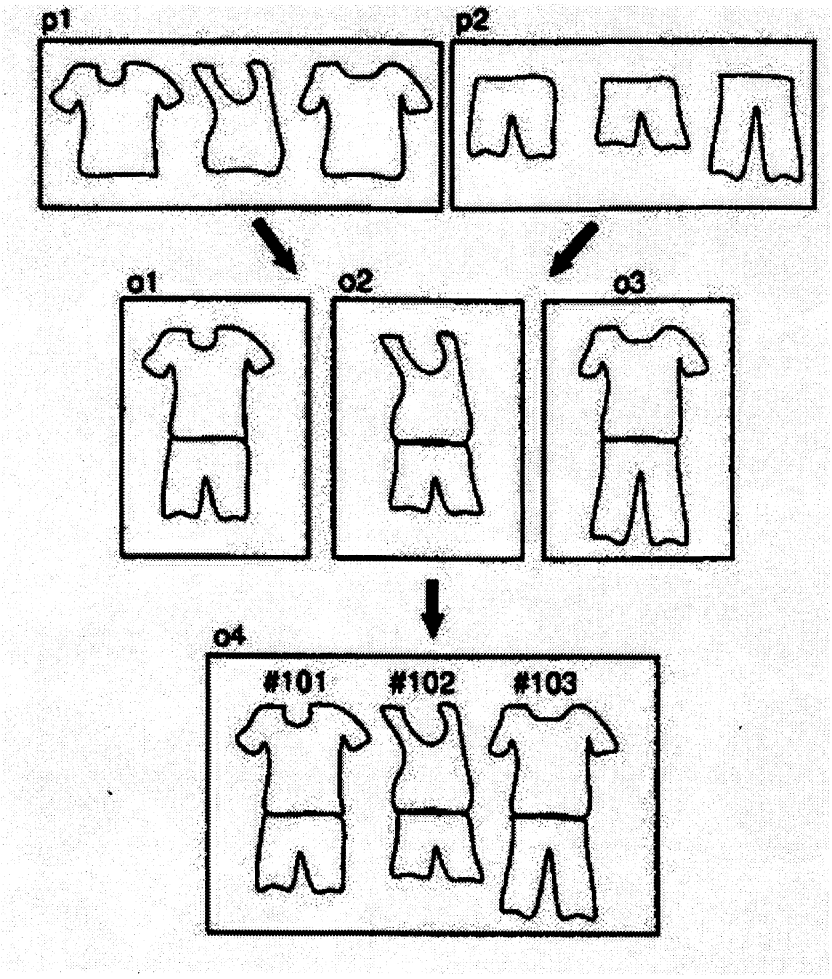

Figure 8. Virtual multimedia objects.

The Virtual Multimedia Object (VMO) [Hoch98] is virtually created based upon other multimedia objects. A pointing entry (a kind of virtual

link) represents the mapping between a VMO and its original data. It can be implemented as user-defined deriving procedures.

A prototype system has been constructed under Solaris 2.5.1 in C++. The system has two layers: Virtual Object layer and Storage layer. Virtual Object Layer manages VMO and their types. Storage layer has the responsibility for storing objects. The technique can be applied to any media type.

6. MULTIMEDIA SOFTWARE PROJECT EFFORT MEASUREMENT

In Section 1 of this chapter we discussed how multimedia technology is utilized in software engineering project management. In this section we discuss how to apply software engineering technology in multimedia project management, and address the issue of multimedia software project effort measurement.

Why multimedia software project effort measurement is an issue? Can't multimedia projects be treated in the same way as other types of software projects? To answer this question, we note that there have been substantial efforts in measuring traditional transaction processing & process control systems, but relatively little effort in determining and evaluating multimedia effort measurements. In fact multimedia software is unique in its emphasis on contents and the need for storyboarding early in the development cycle. Because of that, we not only need tools and techniques specific to multimedia, but also should emphasize different aspects in software project management. For one, managers from diverse background are needed for such projects, to form cross-disciplinary teams. For another, due to the inherently different nature of multimedia systems, the direct application of existing models and measures may not meet multimedia project management needs.

A preliminary empirical study to develop an algorithmic mapping to 'effort' using product characteristics appropriate to multimedia software systems other than LOC or external files is reported in [Fletc98]. The development effort is regarded as a function of 1) building the system content - each media form might have a different impact on development effort - including file name, media type, creation effort, digitizing effort, editing effort, and 2) Authoring the system - a screen that incorporates a greater number of objects and events would take proportionally greater effort to develop. The empirical study on projects developed by 4th year students and delivered in two stages: prototype and final delivery. The students are information science students focussed on programming and design students concentrated on content and interface development. Forty five observations

were recorded in media component development, but no correlation between development effort and either of the component variables - media type and media status. Since data sets were small, generalizations are not possible from this study.

There is a lack of industry driven determination of important attributes. Moreover, data collection needs to be exploratory and student projects are not representative of industry developments. Thus there is a need to develop an industry-based metrics framework in order to determine system and component characteristics considered influential in multimedia software system development efforts. The Goal/Question/Metric (GQM) model shown in Figure 9 is one such framework.

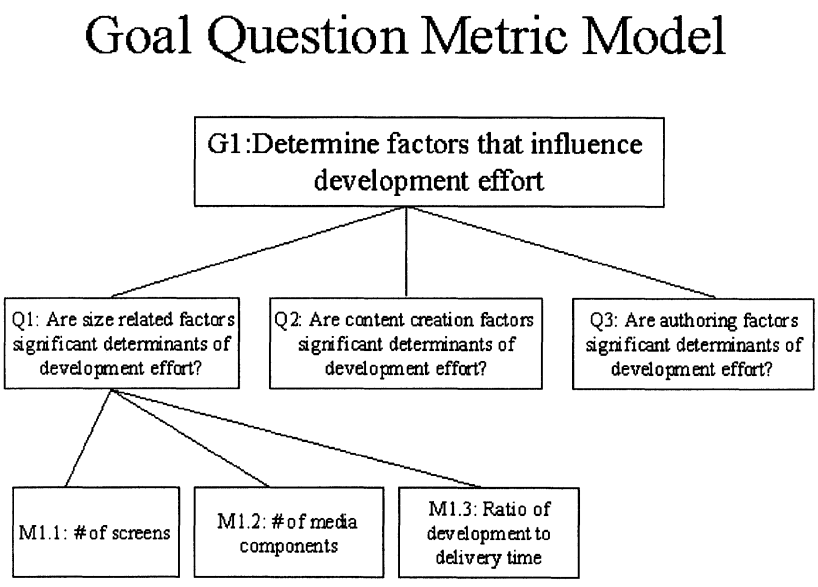

Figure 9. The Goal/Question/Metric Model (a small portion is shown for illustrative purpose).

The key issues considered to reflect industry perspective include:

- Development Tools: Authoring tools and other more complex tools offering visually based high productivity development languages or high level scripting languages.
- Delivery Platform: Platform dictates media format and optimization considerations, for example, CD-I must conform to PAL or NTSC.

- Content Development: Media content development is labor intensive, but difficult to quantify. It depends on media mix, the number of components, and complexity of each component. Artistic considerations also play a role in satisfactory completion of project.
- Organizational Capability: Indirect effects such as size, personnel mix, etc.
- Personnel: Wide variety of skills is required in successful multimedia software development. The core members are a producer, programmer(s), and graphic artist(s).

To perform preliminary verification of the above framework, a pilot study was conducted [MacDo98]. Structured interviews were conducted with three multimedia development organizations. Results suggest the type of project almost entirely determines development environment and workload. Results of pilot study were combined with components of GQM framework to develop a postal survey. The focus is to determine factors that influence development efforts. It was found that fifty per cent of respondents use no formal methodology. None of the traditional software metrics is used. Rather, experience from previous projects is used. Project tracking is performed at very high level, indicating immature project management. Staff experience and project size are obviously important factors.

To summarize, 1) traditional software metrics such as COCOMO and FPA are usually not utilized in multimedia software development; 2) pilot study and questionnaire reinforce the non-formal approach to multimedia software development; 3) although there are attempts to formalize the approach to software development, there is uncertainty on how to achieve this. In other words, multimedia software engineering as a scientific discipline is still evolving, making it an exciting research area to explore.

Chapter 3

Syntax: Visual Languages

As mentioned in Chapter 1, a multimedia application is constructed from a collection of multimedia objects. The primitive objects are media objects of the same media type. The complex multimedia objects are composed from these primitive objects and in general are of mixed media types. The syntax of a multidimensional language ML describes how the complex multimedia objects are constructed from the other multimedia objects. Spatial and temporal composition rules must be taken into consideration.

Before we consider such a multidimensional language for multimedia objects, we may begin with a visual language for visual objects. For one thing, there has been considerable research on visual languages. For another, many multimedia applications are still primarily oriented towards visual objects.

A visual language is a pictorial representation of conceptual entities and operations and is essentially a tool through which users compose iconic, or visual, sentences [Chang95b]. The icons generally refer to the physical image of an object. Compilers for visual languages must interpret visual sentences and translate them into a form that leads to the execution of the intended task [Chang90]. This process is not straightforward. The compiler cannot determine the meaning of the visual sentence simply by looking at the icons. It must also consider the context of the sentence, how the objects relate to one another. Keeping the user's intent and the machine's interpretation the same is one of the most important tasks of a visual language [Crimi90].

1. ICONS

A visual sentence is a spatial arrangement of object icons and/or operation icons that usually describes a complex conceptual entity or a sequence of operations. *Object icons* represent conceptual entities or groups of object icons that are arranged in a particular way. *Operation icons*, also called process icons, denote operations and are usually context-dependent. Figure 1 illustrates a visual sentence that consists of horizontally arranged icons, with a dialog box overlaid on it. This particular location-sensitive visual sentence changes meaning when the locations of icons change, and can be used to specify to-do items for TimeMan, a time-management personal digital assistant. Figure 2 illustrates a content-sensitive visual sentence for TimeMan. The fish in the tank are object icons, each of which represents a to-do item, and the cat is an operation icon that appears when there are too many fish in the tank (the to-do list is too long). On the other hand, Figure 4 in Chapter 4 illustrates a time-sensitive visual sentence that changes its meaning with time.

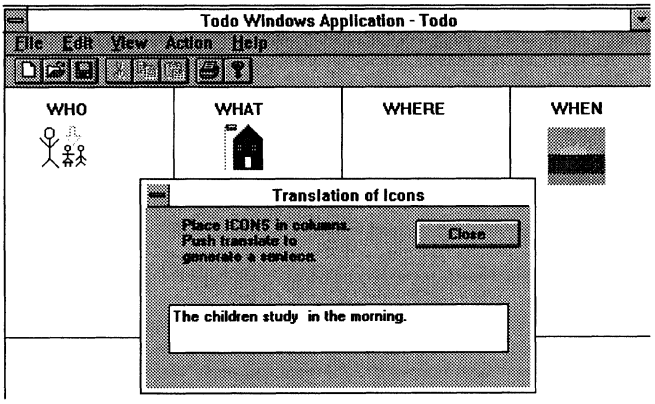

Figure 1. This figure is a location-sensitive visual sentence that shows how the placement of the "school" icon changes its meaning. Here the meaning is "The children study in the morning." In Figure 2 of Chapter 4, the meaning is "The children drive to school in the morning." Such visual sentences can be used to specify to-do items for the time management personal digital assistant TimeMan.

2. OPERATORS

Icons are combined using *operators.* The general form of binary operations is expressed as x_1 *op* $x_2 = x_3$, where the two icons x_1 and x_2 are combined into x_3 using operator *op*. The operator *op* = (op_m, op_p), where op_m is the logical operator, and op_p is the physical operator. Using this expanded notation, we can write (x_{m1}, x_{p1}) *op* (x_{m2}, x_{p2}) = ((x_{m1} op_m x_{m2}), (x_{p1} op_m x_{p2})). In other words, the meaning part x_{m1} and x_{m2} are combined using the logical operator op_m, and the physical part x_{p1} and x_{p2} are combined using the physical operator op_p.

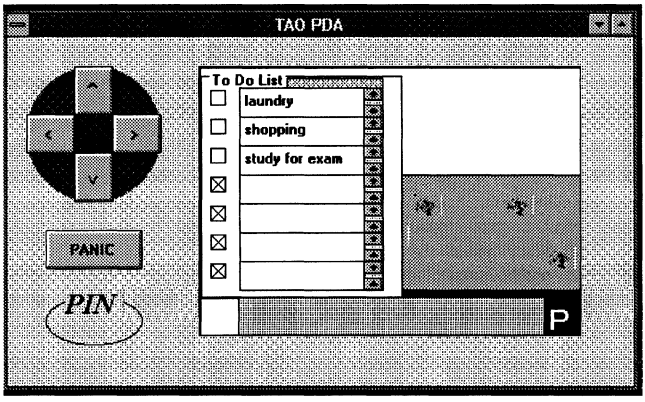

Figure 2. Content-Sensitive visual sentence shows the fish-tank-and-cat metaphor for the time management personal digital assistant TimeMan. Each fish represents a to-do item.

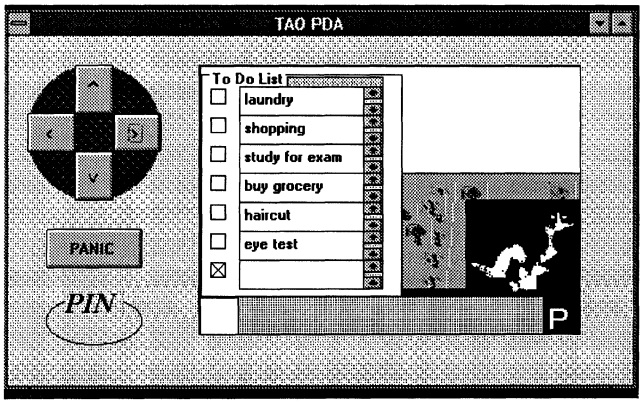

Figure 3. When the to-do list grows too long, the fish tank is overpopulated and the cat appears. The fish tank icon and cat operation icon have corresponding index cells receiving messages from these icons when they are changed by the user (see Chapter 4).

Operators can be *visible* or *invisible*. Most system-defined spatial/temporal operators are invisible, while all user-defined operators are visible for the convenience of the user. For example, excluding the dialog box, the visual sentence in Figure 1 is the horizontal combination of three icons. Therefore, it can be expressed as:

(CHILDREN *hor* SCHOOL_HOUSE) *hor* SUNRISE

where *hor* is an invisible operator denoting a horizontal combination. But if we look at Figure 2, the *cat* is a visible operator denoting a process to be applied to the fish in the fish tank. An operation icon can be regarded as a visible operator.

The four most useful domain-independent icon operators are *ver*, for vertical composition; *hor*, for horizontal composition; *ovl*, for overlay; and *con*, for connect. *ver, hor* and *ovl* are usually invisible, and *con* is usually visible as a connecting line.

The invisible icon operators are *spatial operators* and apply only to icons or ticons. The spatial composition of two icons or ticons is a *complex icon*.

3. REPRESENTING MEANING

To represent the meaning of an icon, we use either a frame or a conceptual graph, depending on the underlying semantic model of the application system being developed. Both are appropriate representations of meaning, and can be transformed into one another. For example, the SCHOOL_HOUSE icon in Figure 1 can be represented by the following frame:

Icon SCHOOL_HOUSE
WHO: nil
DO: study
WHERE: school
WHEN: nil

In other words, the SCHOOL_HOUSE icon has the meaning "study" if it is in the DO location, or the meaning "school" in the WHERE location. Its meaning is "nil" if it is in the WHO or WHEN location. An equivalent linearized conceptual graph is as follows:

[Icon = SCHOOL_HOUSE]
--(sub)--> [WHO = nil]
--(verb)-> [DO = study]
--(loc)--> [WHERE = school]
--(time)-> [WHEN = nil]

The meaning of a composite icon can be derived from the constituent icons, if we have the appropriate inference rules to combine the meanings of the constituent icons. We have applied conceptual dependency theory to develop inference rules to combine frames [Chang94b]. We have also adopted conceptual operators to combine conceptual graphs [Chang89]. As a simple example, the merging of the frames for the icons in the visual sentence shown in Figure 1 will yield the frame:

Visual_Sentence vs_1
WHO: children
DO: study
WHERE: nil
WHEN: morning

We can derive this frame by merging the frames of the four icons using the following rule:

The i^{th} slot gets the value of the corresponding slot of the i^{th} icon.

Thus the first slot with slot_name WHO gets the value “children” from the corresponding slot of the first icon CHILDREN, the second slot with slot_name DO gets the value "study" from the corresponding slot of the second icon SCHOOL_HOUSE, etc.

Chapter 4

Syntax: Multimedia Languages

Visual languages, which let users customize iconic sentences, can be extended to accommodate multimedia objects, letting users access media dynamically. Teleaction objects, or multimedia objects with knowledge structures, can be designed using visual languages to automatically respond to events and perform tasks like "find related books" in virtual library Bookman.

Languages that let users create custom icons and iconic sentences are receiving increased attention, as multimedia applications become more prevalent. Visual language systems let the user introduce new icons, and create iconic sentences with different meanings and the ability to exhibit dynamic behavior. With a graphical user interface, the user must generally compose commands with predefined icons, which limits the range of commands and makes dynamic composition rather difficult. It is also awkward to customize such commands without cluttering the screen. These limitations are significant in multimedia applications because the user must often access multimedia information dynamically with very few icons.

At the University of Pittsburgh and Knowledge Systems Institute, we have developed a formal framework for visual language semantics that is based on the notion of icon algebra and have designed several visual languages for the speech impaired. In Chapter 3 we described the underlying grammar for a visual language. We have since extended the framework to include the design of *multidimensional languages* — languages that capture the dynamic nature of multimedia objects through icons, earcons (sound), micons (motion icons), and vicons (video icons). The user can create a multimedia message by combining these icons and have direct access to multimedia information, including animation.

We have successfully implemented this framework in developing Bookman, an interface to a virtual library used by the students and faculty of the Knowledge Systems Institute. As part of this work, we extended the visual language concepts to develop *teleaction objects*, objects that automatically respond to some events or messages to perform certain tasks [ChangH95b]. We are continuing work on extensions to the visual interface in the context of emergency management, where the information system must react to flood warnings, fire warnings, and so on, to present multimedia information and to take actions [Khali96].

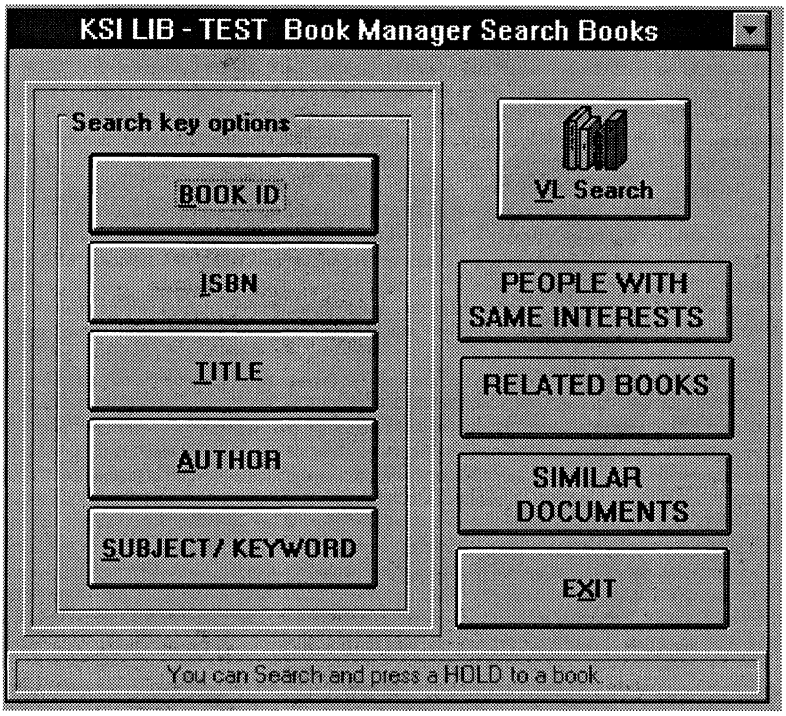

Figure 1. The BookMan interface to a virtual library lets the user select different search modes.

Figure 1 shows the search and query flexibility possible with the Bookman interface. In addition, users can perform a range of tasks, including finding related books, finding books containing documents similar to documents contained in the current book, receiving alert messages when related books or books containing similar documents have been prefetched by BookMan, finding other users with similar interests or receiving alert messages about such users (the last function requires mutual consent among the users) etc. In developing the interface, our goal was to give users the

same range of freedom they might experience in a real library. Much of this power stems from the use of TAOs.

1. WHAT TELEACTION OBJECTS DO

To create a TAO, we attached knowledge about events to the structure of each *multimedia object* — a complex object that comprises some combination of text, image, graphics, video, and audio objects. TAOs are extremely valuable because they greatly improve the selective access and presentation of relevant multimedia information. In BookMan, for example, each book or multimedia document is a TAO because the user can not only access the book, browse its table of contents, read its abstract, and decide whether to check it out, but also be informed about related books, or find out who has a similar interest in this subject. The user can indicate an intention by incrementally modifying the physical appearance of the book, usually with just a few clicks of the mouse.

TAOs can accommodate an almost limitless range of functions. For example, when the user clicks on a particular book, it can automatically access information about related books and create a multimedia presentation from all the books.

The drawback of TAOs is that they are complex objects and therefore the end user can not easily manipulate them with traditional define, insert, delete, modify, and update commands. Instead, TAOs require direct manipulation, which we provided through a multidimensional language.

The physical appearance of a TAO is described by a *multidimensional sentence*. The syntactic structure derived from this multidimensional sentence controls its dynamic multimedia presentation. The TAO also has a knowledge structure called the *active index* that controls its event-driven or message-driven behavior. The multidimensional sentence may be location-sensitive, time-sensitive or content-sensitive. Thus, an incremental change in the TAO's external appearance is an event that causes the active index to react. As I describe later, the active index itself can be designed using a visual-language approach.

2. MULTIDIMENSIONAL LANGUAGE

The multidimensional language consists of generalized icons and operators, and each sentence has a syntactic structure that controls the dynamics of a multimedia presentation.

2.1 Generalized icons and operators

In Chapter 3 we presented the elements of visual languages and described the icons and operators in a visual (not multidimensional) language. In a multidimensional language, we want not only icons that represent objects by images, but also icons that represent the different types of media. We call such primitives *generalized icons* and define them as $x = (x_m, x_p)$ where x_m is the meaning and x_p is the physical appearance. To represent TAOs, we replace the x_p with other expressions that depend on the media type:

- Icon: (x_m, x_i) where x_i is an image
- Earcon: (x_m, x_e) where x_e is sound
- Micon: (x_m, x_s) where x_s is a sequence of icon images (motion icon)
- Ticon: (x_m, x_t) where x_t is text (ticon can be regarded as a subtype of icon)
- Vicon: (x_m, x_v) where x_v is a video clip (video icon)

The combination of an icon and an earcon/micon/ticon/vicon is a multidimensional sentence.

For multimedia TAOs, we define operators as

- Icon operator $op = (op_m, op_i)$, such as *ver* (vertical composition), *hor* (horizontal composition), *ovl* (overlay), *con* (connect), *surround*, *edge_to_edge*, etc.
- Earcon operator $op = (op_m, op_e)$, such as *fade_in, fade_out*, etc.
- Micon operator $op = (op_m, op_s)$, such as *zoom_in, zoom_out*, etc.
- Ticon operator $op = (op_m, op_t)$, such as *text_merge, text_collate*, etc.
- Vicon operator $op = (op_m, op_v)$, such as *montage, cut*, etc.

Two classes of operators are possible in constructing a multimedia object. As described in Chapter 3, spatial operators are operators that involve spatial relations among image, text or other spatial objects. A multimedia object can also be constructed using operators that consider the passage of time. *Temporal operators,* which apply to earcons, micons, and vicons, make it possible to define the temporal relation [Allen83] among generalized icons. For example, if you want to watch a video clip and at the same time listen to the audio, you can request that the video *co_start* with the audio. Temporal operators for earcons, micons, ticons and vicons include *co_start, co_end, overlap, equal, before, meet,* and *during* and are usually treated as invisible operators because they are not visible in the multidimensional sentence.

When you use temporal operators to combine generalized icons, their types may change. For example, a micon followed in time by another icon is still a micon, but the temporal composition of micon and earcon yields a vicon. Media type changes are useful in *adaptive multimedia* so that one type of media may be replaced/combined/augmented by another type of media (or a mixture of media) for people with different sensory capabilities.

We can add still more restrictions to create subsets of rules for icons, earcons, micons and vicons that involve *special operators*:

- For earcons, special operators include *fade_in, fade_out,*
- For micons, special operators include *zoom_in, zoom_out*,
- For ticons, special operators include *text_collate, text_merge*,
- For vicons, special operators include *montage, cut.*

These special operators support the combination of various types of generalized icons, which means the multidimensional language can fully reflect all multimedia types.

2.2 Grammar

Multidimensional languages can handle temporal as well as spatial operators. A visual language has a relational grammar, *G*, which a compiler uses to generate sentences:

$$G = (N, X, OP, s, R)$$

where N is the set of nonterminals, X is the set of terminals (icons), OP is the set of spatial relational operators, s is the start symbol, and R is the set of production rules whose right side must be an expression involving relational operators.

To describe multidimensional languages, we extended the X and OP elements of G: X is still the set of terminals but now includes earcons, micons, ticons, and vicons as well as icons, and the OP set now includes temporal as well as spatial relational operators.

2.3 Syntax

Informally, a multidimensional language is a set of multidimensional sentences, each of which is the spatial/temporal composition of generalized icons. Figure 2 without the dialog box illustrates a simple visual sentence, which describes the physical appearance of a multimedia object retrieved by BookMan. With the dialogue box, the figure becomes a multidimensional

sentence used by BookMan to generate "The children drive to school in the morning." in synthesized speech. The multidimensional sentence has the syntactic structure

(DIALOG_BOX *co_start* SPEECH) *ver* (((CHILDREN *hor* CAR) *hor* SCHOOL_HOUSE) *hor* SUNRISE)

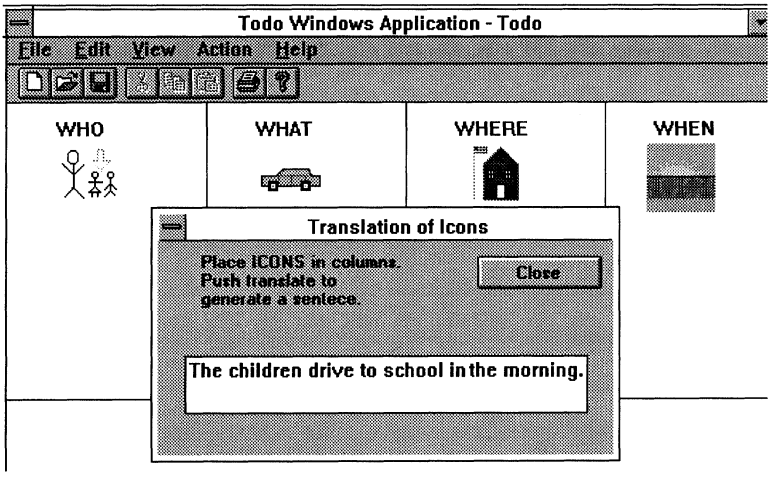

Figure 2. A multidimensional sentence whose meaning changes when the icons change their positions and is therefore a location-sensitive sentence. This sentence has the meaning "The children drive to school in the morning."

Figure 3 is a hypergraph of the syntactic structure. The syntactic structure is essentially a tree, but it has additional temporal operators (such as *co_start*) and spatial operators (such as *hor* and *ver*) indicated by dotted lines. Some operators may have more than two operands (for example, the *co_start* of audio, image, and text), which is why the structure is called a hypergraph. The syntactic structure controls the multimedia presentation of the TAO.

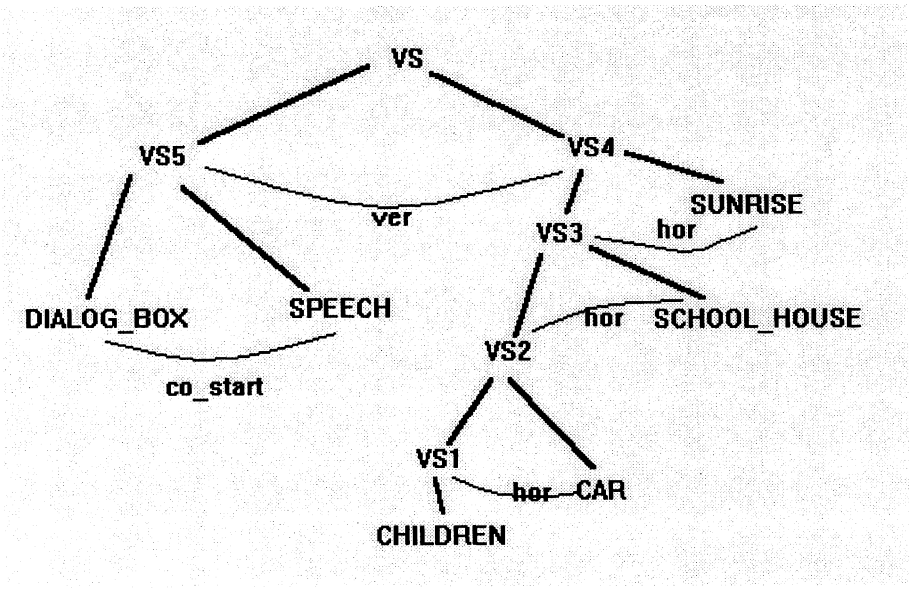

Figure 3. The syntactic structure of the multidimensional sentence shown in Figure 2. This structure is a hypergraph because some relational operators may correspond to lines with more than two end points.

Multidimensional languages must also account for multimedia dynamics because many media types vary with time. This means that a dynamic multidimensional sentence changes over time.

We defined rules for spatial and temporal operators that let us transform the hypergraph in Figure 3 to a Petri net that controls the multimedia presentation. Figure 4 represents the Petri net of the sentence in Figure 2. As such, it also a representation of the dynamics of the multidimensional sentence in Figure 2. The multimedia presentation manager can execute this Petri net dynamically to create a multimedia presentation [Lin96]. For example, the presentation manager will produce the visual sentence in Figure 2 as well as the synthesized speech.

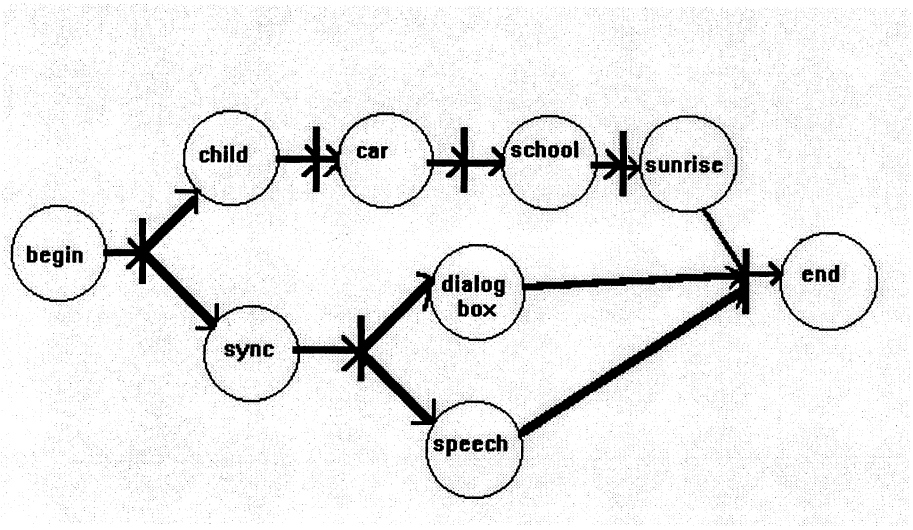

Figure 4. A time-sensitive visual sentence for the Petri net controlling the presentation of the multidimensional sentence shown in Figure 2.

3. KNOWLEDGE STRUCTURE

An *index cell* is the fundamental building block of the active index, which is the key element of a TAO [Chang95a]. Without the active index, a TAO would not be able to react to events or messages, and the dynamic visual language would lose its power.

3.1 Cell communication

An index cell accepts input messages, performs some action, and posts an output message to a group of output index cells. Depending on its internal state and the input messages, the index cell can post different messages to different groups of output index cells. Therefore the connection between an index cell and its output cells is dynamic. For example, if a Bookman user wants to know about new books on nuclear winter, he modifies the visual sentence, causing TAO to send a message to activate a new index cell that will collect information on nuclear winter.

An index cell can be either live or dead, depending on its internal state. The cell is live if the internal state is anything but the dead state. If the internal state is the dead state, the cell is dead. The entire collection of index

cells, either live or dead, forms the *index cell base*. The set of live cells in the index cell base forms the active index.

Each cell has a built-in timer that tells it to wait a certain time before deactivating (dead internal state). The timer is re-initialized each time the cell receives a new message and once again becomes active (live). When an index cell posts an output message to a group of output index cells, the output index cells become active. If an output index cell is in a dead state, the posting of the message will change it to the initial state, making it a live cell, and will initialize its timer. On the other hand, if the output index cell is already a live cell, the posting of the message will not affect its current state but will only re-initialize its timer.

Active output index cells may or may not accept the posted message. The first output index cell that accepts the output message will remove this message from the output list of the current cell. (In a race, the outcome is nondeterministic.) If no output index cell accepts the posted output message, the message will stay indefinitely in the output list of the current cell. For example, if no index cells can provide the BookMan user with information about nuclear winter, the requesting message from the nuclear winter index cell will still be with this cell indefinitely.

After its computation, the index cell may remain active (live) or deactivate (die). An index cell may also die if no other index cells (including itself) post messages to it. Thus the nuclear winter index cell in BookMan will die if not used for a long time, but will be re-initialized if someone actually wants such information and sends a message to it.

Occasionally many index cells may be similar. For example, a user may want to attach an index cell to a document that upon detecting a certain feature sends a message to another index cell to prefetch other documents. If there are 10,000 such documents, there can be ten thousand similar index cells. The user can group these cells into an *index cell type*, with the individual cells as instances of that type. Therefore, although many index cells may be created, only a few index cell types need to be designed for a given application, thus simplifying the application designer's task.

3.2 Cell construction

To aid multimedia application designers in constructing index cells, we developed a visual-language-based tool, IC Builder, and used it to construct the index cells for the BookMan interface. Figure 5 shows a prefetch index cell being built. Prefetch is used with two other index cell types to retrieve documents [Chang95a]. If the user selects the prefetch mode of the BookMan interface, the active index will activate the links to access information about related books. Prefetch is responsible for scheduling

prefetching, initiating (issuing) a prefetching process to prefetch multimedia objects, and killing the prefetching process when necessary.

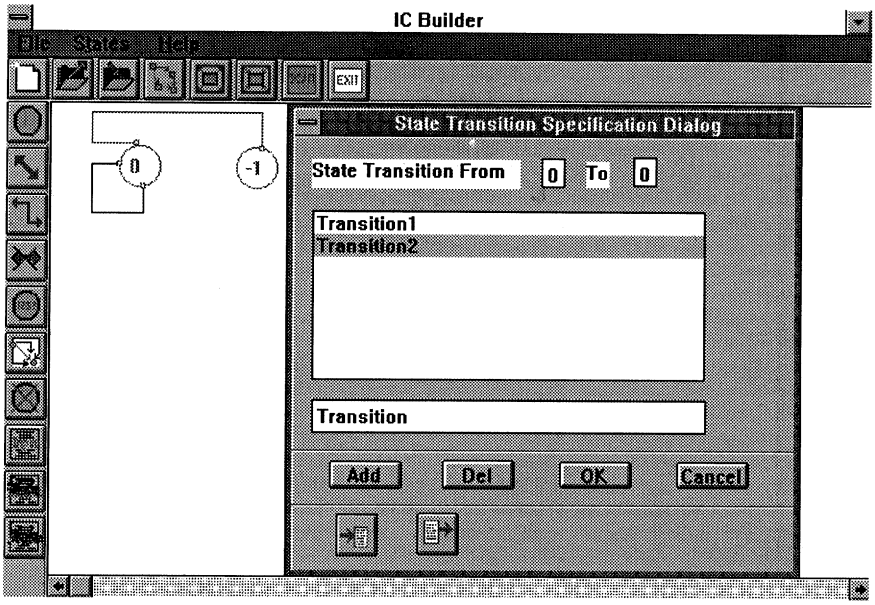

Figure 5. The visual specification of the state transitions for an active index cell of the virtual library's user interface BookMan.

Figure 5 shows the construction of the state-transition diagram. The prefetch index cell has two states: state 0, the initial and live state, and state -1, the dead state. The designer draws the state-transition diagram by clicking on the appropriate icons. In this example, the designer has clicked on the fourth vertical icon (zigzag line) to draw a transition from state 0 to state 0. Although the figure shows only two transition lines, the designer can specify as many transitions as necessary from state 0 to state 0. Each transition could generate a different output message and invoke different actions. For example, the designer can represent different prefetching priority levels in BookMan by drawing different transitions.

The designer wants to specify details about transition2 and so has highlighted it. Figure 6 shows the result of clicking on the input message icon (top icon to the right of the State Transition Specification Dialog box.) IC Builder brings up the Input Message Specification Dialog box so that the designer can specify the input messages. The designer specifies message 1 (start_prefetch) input message. The designer could also specify a predicate, and the input message is accepted only if this predicate is evaluated true. Here there is no predicate, so the input message is always accepted.

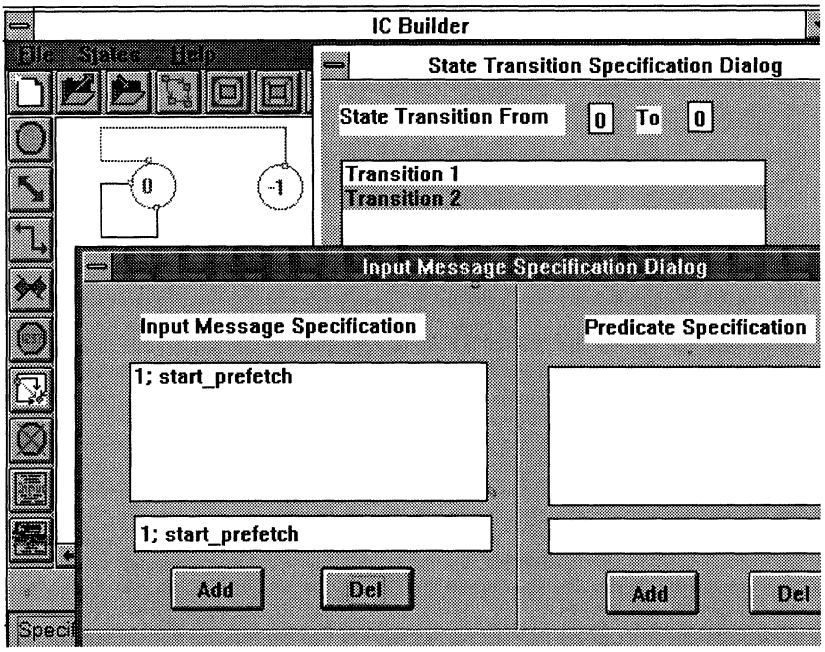

Figure 6. The visual specification of the input message for an active index cell of the virtual library's user interface BookMan.

Figure 7 shows what happens if the designer clicks on the output message icon in Figure 5 (bottom icon to the right of the State Transition Specification Dialog box). IC Builder brings up the Output Message Specification Dialog box so that the designer can specify actions, output messages, and output index cells. In this example, the designer has specified three actions: compute_schedule (determine the priority of prefetching information), issue_prefetch_proc (initiate a prefetch process), and store_pid (Once a prefetch process is issued, its process id or pid is saved so that the process can be killed later if necessary).

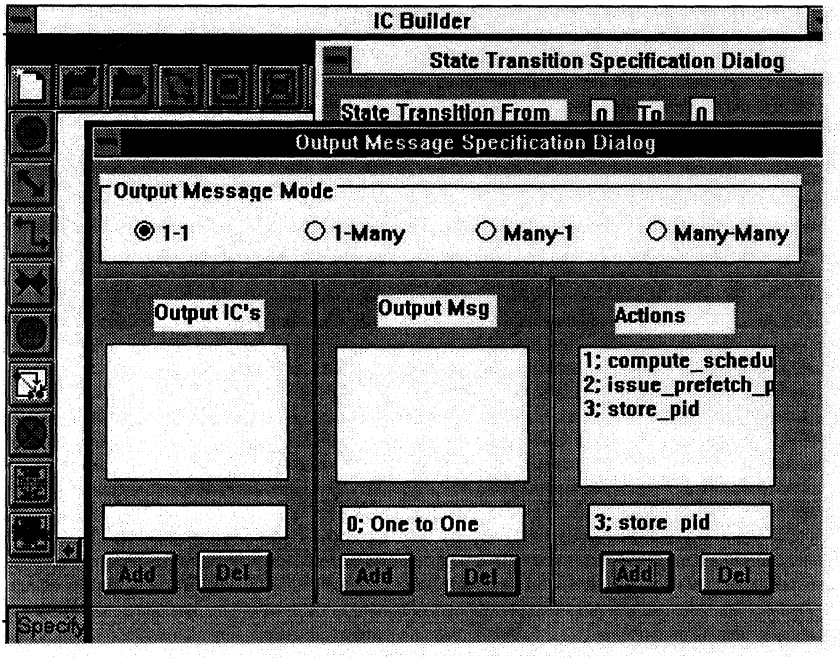

Figure 7. The visual specification of the output message and actions for an active index cell of the virtual library's user interface BookMan.

In the figure, there is no output message, but both input and output messages can have parameters. The index cell derives the output parameters from the input parameters.

4. DYNAMIC VISUAL LANGUAGE FOR QUERYING

When the user makes incremental changes to a multidimensional sentence, certain events occur and messages are sent to the active index. For example, suppose the user clicks on a book TAO to change the color attribute of the book. This is a *select* event, and the message *select* is sent to the active index. If the user creates a new *related_info* operation icon, this is a *related_info* event, and a message *prefetch_related_info* is sent to the

active index. The incremental changes to a multidimensional sentence can be either:

- *Location-sensitive*. The location attribute of a generalized icon is changed.
- *Time-sensitive*. The time attribute of a generalized icon is changed.
- *Content-sensitive*. An attribute of a generalized icon other than a location or time attribute is changed or a generalized icon is added or deleted, or an operator is added or deleted.

A visual sentence or multidimensional sentence can also be location-sensitive, time-sensitive, or content-sensitive. Chapter 3 gives examples of different types of visual sentences. The resulting language is a dynamic visual language or dynamic multidimensional language.

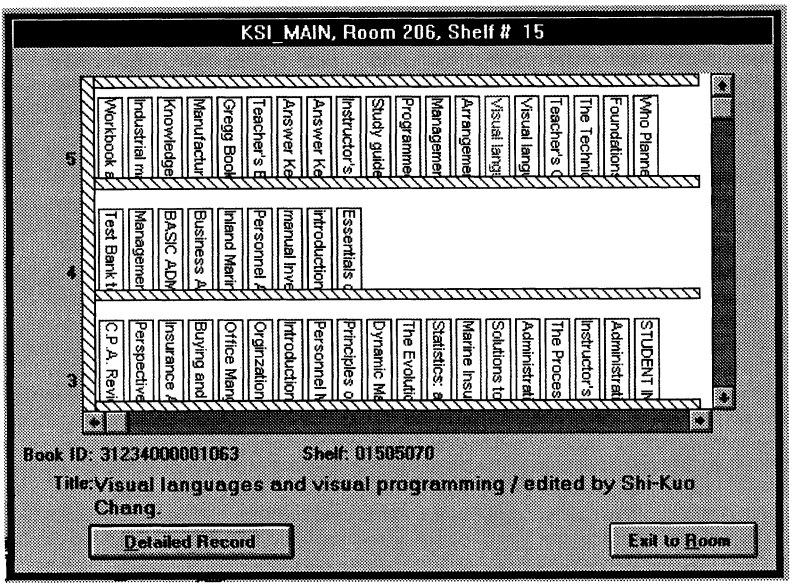

Figure 8. The BookMan interface to a virtual library lets the user browse the virtual library and select desired book for further inspection.

A dynamic visual language for virtual reality serves as a new paradigm in a querying system with multiple paradigms (form-based queries, diagram-based queries and so on) because it lets the user freely switch paradigms [Chang94a]. When the user initially browses the virtual library, the VR query may be more natural; but when the user wants to find out more details,

the form-based query may be more suitable. This freedom to switch back and forth among query paradigms gives the user the best of all worlds, and dynamic querying can be accomplished with greater flexibility.

From the viewpoint of dynamic languages, a VR query is a location-sensitive multidimensional sentence. As Figure 8 shows, BookMan indicates the physical locations of books by marked icons in a graphical presentation of the books stacks of the library. What users see is the same (with some simplification) as what they would experience in a real library. That is, the user selects a book by picking it from the shelf, inspects its contents and browses adjacent books on the shelf.

In Figure 1, initially the user is given the choice of query paradigms: search by title, author, ISBN, or keyword(s). If the user selects the virtual library search, he can then navigate in the virtual library, and as shown in Figure 8, the result is a marked object. If the user switches to a form-based representation by clicking the "DetailedRecord" button, the result is an item in the form of Figure 9. The user can now use the form to find books of interest, and switch back to the VR query paradigm by clicking the "VL location" button in Figure 9.

Essentially, the figure illustrates how the user can switch between a *VR paradigm* (such as the virtual library) and a *logical paradigm* (such as the form).

There are certain admissibility conditions for this switch. For a query in the logical paradigm to be admissible to the VR paradigm, the retrieval target object should also be an object in VR. For example, the virtual reality in the Bookman library is stacks of books, and an admissible query would be a query about books, because the result of that query can be indicated by marked book icons in the virtual library.

Conversely, for a query in the VR paradigm to be admissible to the logical paradigm, there should be a single marked VR object that is also a database object, and the marking is achieved by an operation icon such as *similar_to* (find objects similar to this object), *near* (find objects near this object), *above* (find objects above this object), *below* (find objects below this object), and other spatial operators. For example, in the VR for the virtual library, a book marked by the operation icon *similar_to* is admissible and can be translated into the logical query "find all books similar to this book."

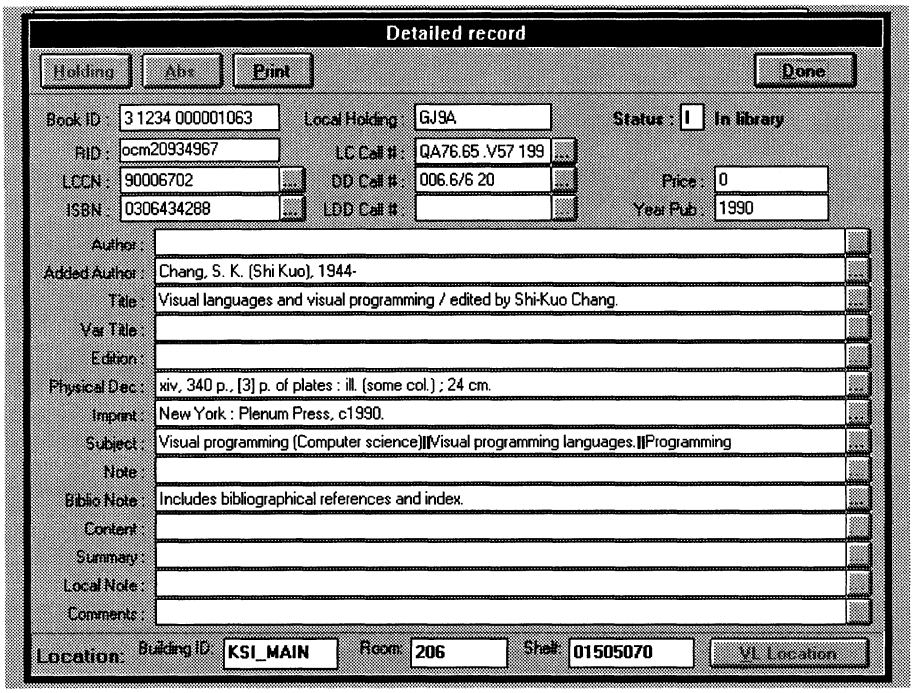

Figure 9. The BookMan interface to a virtual library also lets the user switch to a traditional form-based query mode.

5. DISCUSSION

Visual query systems for multimedia databases, like Bookman, are under active investigation at many universities as well as industrial laboratories. These systems are extremely flexible. For example, a user can easily and quickly ask for any engineering drawing that contains a part that looks like the part in another drawing and that has a signature in the lower right corner that looks like John Doe's signature. In fact, in our work on Bookman, we plan to build a mechanism that will let users create similarity retrieval requests that prompt Bookman to look for books similar to the book being selected.

We have implemented the active index for BookMan to support optional modes like prefetching. We can also extend BookMan to perform searches on the World Wide Web using a Web browser enhanced with an active index [Chang95a].

We have also used our multidimensional language framework to design user interfaces for personal digital assistants. Chapter 3 described TimeMan, a personal assistant that performs time-management functions. Just as books in BookMan are teleaction objects, so are calendars in TimeMan. We used a multidimensional language to describe the external appearance of a TAO calendar, and provided an active index to manage to-do items.

In summary, visual languages and multidimensional languages are useful in specifying the syntactic structure, knowledge structure, and dynamic behavior of complex multimedia objects such as TAO. As multimedia applications become widespread, we expect to see more visual query systems in which multidimensional languages will play an important role, both as a theoretical foundation and as a means to explore new applications.

Chapter 5

Semantics: The Active Index

In multimedia computing, an important issue is how to index multimedia objects, so that the multimedia objects can be accessed quickly and certain actions can be performed automatically. In conventional database systems, keyword-based indexing techniques are adequate to support users' needs. In multimedia information systems, there are many applications that cannot be properly supported by keyword-based techniques. In addition to keywords, users often want to access/manipulate multimedia objects by shape, texture, spatial relationships, etc. That is, certain features of the multimedia objects are used as indexes, and, in many cases, they cannot be represented as keywords. The representation of these feature-based indexes poses some special problems:

(1) Indexes are approximately represented.

(2) Indexes do not have an implicit ordering, in the sense that if **a**, **b** and **c** are three index values and **a** < **b** < **c**, it does not mean that multimedia object b is more similar to visual object **a** than multimedia object **c**.

(3) Indexes may have interrelated multiple attributes.

Faced with these problems, the conventional indexing structures such as B-tree, hashing, etc. cannot be used for the organization of indexes for multimedia objects. New indexing structures must be explored which should also support similarity retrieval. Moreover, the index structures should be highly flexible and dynamic, with the following characteristics:

(a) Active index instead of passive index: The index can be used to perform actions.

(b) Partial index instead of total index: Only a few multimedia objects are indexed.

(c) Dynamic index instead of static index: The index can evolve, grow and shrink.

(d) Visible index instead of transparent index: The user is aware of the existence of the index, perhaps as part of the knowledge structure. So the index is not necessarily transparent.

(e) Imprecise index instead of precise index: The index can be used in processing imprecise or approximate queries.

This chapter introduces a theoretical framework for the active index. The theoretical framework is introduced in Section 1. To illustrate its application to the Smart Image System, in Section 2 we present a three-level active index. With an active index, we can effectively and efficiently handle smart images that can respond to accessing, probing, and other actions. The application to information retrieval in hyperspace is discussed in Section 3. The computation power of the active index is analyzed in Section 4. The active index can be used to realize Petri nets, generalized Petri nets such as G-nets, B-trees, etc., but the dynamic nature of the active index makes it even more powerful and flexible. The reversible index introduced in Section 5 facilitates feature-based indexing. An experimental active index system has been implemented, whose main features are described in Section 6. In Section 7, further research topics are discussed.

1. FORMAL DEFINITION OF THE ACTIVE INDEX

An index cell base (ICB) consists of a (possibly infinite) number of index cells. An index cell (ic) accepts input messages and performs some computation. It then activates another group of index cells, and posts the output message to these output index cells. If some of these output index cells have already been activated, they may simply accept the output from the current index cell. The first output cell that accepts the output message will remove it from the output list of the current cell.

After its computation, the index cell may remain active (live), or de-activate itself (dead). An index cell will also become dead, if it remains

inactive for a certain period of time, i.e., if no other cells (including itself) send messages to it.

An active index (IX) consists of a finite number of index cells ic from ICB. Thus an active index IX is a finite subset of the (possibly infinite) index cell base ICB. When the active index is in actual computation, it consists of a time-varying collection of index cells in different states, accepting certain input messages and posting output messages to the output lists. To describe precisely the behavior of the active index, we will first formally define an index cell.

Definition 1: An index cell is described by ic = (X, Y, S, s_o, A, t_{max}, f, g) where:

X is the set of input messages including dummy input d.

Y is the set of output messages including dummy output d.

S is the set of states. S includes a set of ordinary states S and a special state s_{dead} called the dead state. If an index cell is in the dead state, it is a dead index cell. Otherwise it is a live index cell.

s_o in S is the initial state of the index cell ic. A is the set of action sequences that can be performed by this index cell.

t_{max} is the maximum time for the cell to remain live, without receiving any messages. If t_{max} is infinite, the cell is perennial.

f is a function: 2^X x S -> {0,1} where 2^X is the power set of input X. If f({x_1 ,..., x_m}, s) is 1, then the cell accepts the input set { x_1 ,..., x_m} and x_1,..., x_m are removed from the output lists of those cells that produce these output messages. The removal of messages is an atomic action that will occur simultaneously. If f({x_1,..., x_m }, s) is 0, the input messages are not accepted. When several input sets can be accepted, one is chosen non-deterministically.

g is a function: 2^X x S -> 2^{ICB} x Y x S x A such that given input messages {x_1 ,..., x_m} which have been accepted, i.e., f({x_1,..., x_m},s) = 1, and current state s, g(x,s) is a quadruple (Ic, y, s', a) where

(1) Ic is a set of output index cells to be activated. If an output index cell is in the dead state, it is changed to the initial state so that it becomes a live cell, and the clock t is initialized to be t_{max}. If an output index cell is already live, its current state remains unchanged, but its clock t is re-initialized to be t_{max}. If an output index cell is the special symbol nil, no output index cell is actually activated.

(2) y is the output message for the output index cells in Ic. The output could be the dummy message d, when there is no real output to the output index cells. The first output index cell that accepts this output message y will remove it from the output list of ic.

(3) s' is the computed next state of ic. The true next state s of ic is the dead state if clock time t becomes zero or negative, and s' otherwise. If the next state s' is the dead state, the index cell becomes dead.

(4) a is the action-sequence performed by this index cell, which can be regarded as the output of the cell to the external environment.

Definition 2: The output list ***oL*** of an ic is of the following form: [(Ic_1, y_1), (Ic_2, y_2), ..., (Ic_m, y_m)], where y_i is the output message posted to the ic's in the set Ic_i. If any ic in Ic_i accepts y_i, the tuple (Ic_i, y_i) is removed from the output list.

Definition 3: An index cell base ICB is a (possibly infinite) collection of index cells. Given an index cell base ICB, an active index IX is a finite subset of ICB with n index cells, denoted by an n-place ic vector ***ic*** = (ic_1, ic_2, ..., ic_n), where the ic_i's are ordered by their (arbitrary) subscripts in ICB.

Definition 4: The instantaneous description id of an an active index IX is denoted by id = (***ic***, ***s***, ***oL***), where ***ic*** is the ic vector, ***s*** is the corresponding state vector, and ***oL*** is the corresponding output list vector.

Definition 5: The trace of an active index IX with respect to ($\boldsymbol{ic}_0$, $\boldsymbol{s}_0$, $\boldsymbol{oL}_0$) is:

($\boldsymbol{ic}_0$, $\boldsymbol{s}_0$, $\boldsymbol{oL}_0$) $\Rightarrow$
($\boldsymbol{ic}_1$, $\boldsymbol{s}_1$, $\boldsymbol{oL}_1$) $\Rightarrow$
.....
($\boldsymbol{ic}_n$, $\boldsymbol{s}_n$, $\boldsymbol{oL}_n$)

where ($\boldsymbol{ic}_i$, $\boldsymbol{s}_i$, $\boldsymbol{oL}_i$) $\Rightarrow$ ($\boldsymbol{ic}_{I+1}$, $\boldsymbol{s}_{I+1}$, $\boldsymbol{oL}_{I+1}$) due to the acceptance of input messages by an index cell. The $\Rightarrow$ symbol reads as "*is transformed into*".

Such transformations may occur in any arbitrary order. Each transformation step in the trace takes exactly one clock cycle.

If the trace is finite, the active index IX is terminating with respect to ($\boldsymbol{ic}_0$, $\boldsymbol{s}_0$, $\boldsymbol{oL}_0$); otherwise it is nonterminating.

Therefore, an active index is initially specified by ($\boldsymbol{ic}_0$, $\boldsymbol{s}_0$, $\boldsymbol{oL}_0$) where $\boldsymbol{ic}_0$ is the initial ic vector, $\boldsymbol{s}_0$ is the initial state vector and $\boldsymbol{oL}_0$ is the initial output list vector.

For example, we can start with a single index cell, so that the active index initially starts with ($\boldsymbol{ic}_0$, $\boldsymbol{s}_0$, $\boldsymbol{oL}_0$), where $\boldsymbol{s}_0$ is the initial state of $\boldsymbol{ic}_0$, and the output list $\boldsymbol{oL}_0$ is empty. Let f({}, $\boldsymbol{s}_0$) be 1, so that $\boldsymbol{ic}_0$ will accept an empty set as input. Let f(V, $\boldsymbol{s}_0$) be 0 for any non-empty V. If we intend to activate index cells in Ic and post a message y to them, it can be done by an appropriate g function. After that, $\boldsymbol{ic}_0$ enters a state s_{sleep}, where no input will

be accepted. The t_{max} can be set to infinity. In other words, the sole purpose of ic_0 is to activate some index cells and post a message to them. By adding states to ic_0 appropriately, we can also make ic_0 post individual messages to each of the activated index cells.

Observation 1: An index cell ic can be modified to post n messages individually to n output index cells.

Proof: Suppose $f(\{x_1, ..., x_m\}, s) = 1$ and we want to post messages y_i individually to ic_i, $1 \leq i \leq n$, and then change state to s'. Let $s_1, s_2, ..., s_{n-1}$ be n-1 new states. Replace the original $g(\{x_1,..., x_m\}, s) = (Ic, y, s', \mathbf{a})$ by the following: $g'(\{x_1,..., x_m\}, s) = (\{ic_1\}, y_1, s_1, \mathbf{a})$.

We can construct f' and g' as follows:

$f'(\{\}, s_1) = 1$ and $f'(\{\}, s'') = 0$ if $s'' \neq s_1$
$g'(\{\}, s_1) = (\{ic_2\}, y_2, s_2, nil)$
$f'(\{\}, s_2) = 1$ and $f'(\{\}, s'') = 0$ if $s'' \neq s_2$
$g'(\{\},s_2) = (\{ic_3\}, y_3, s_3, nil)$
....
$f'(\{\}, s_{n-1}) = 1$ and $f'(\{\}, s'') = 0$ if $s'' \neq s_{n-1}$
$g'(\{\},s_{n-1}) = (\{ic_n\}, y_n, s', nil)$

Therefore, the ic will go through the states $s_1, ..., s_{n-1}$ and post the messages individually to the output ic's, and then change state to s'. ■

Notation 1: As a notational convenience, we will write the quadruple as (Ic, W, s', a), where the cardinality of Ic and W must be identical, i.e., $Ic = \{ic_1, ..., ic_n\}$, $W = \{y_1, ..., y_n\}$, to indicate that each y_i is posted to each ic_i individually.

Notation 2: As a further notational convenience, we will allow Ic and W to be lists. If Ic is of the form [ic,...,ic], this means the messages y_i are all posted to the same ic. If W is of the form [y,...,y], this means the same message y is posted to each ic_i individually.

The above notation enables us to specify the posting of a message y either to an individual ic, or to a group of ic's. In particular, a message can be posted to a certain type of ic, if we do not yet know the identity of the individual ic.

The external environment may also send messages to the active index. In particular, the action sequence may cause the external environment to send messages to some of the index cells, including the index cell that performs

the said action sequence. This can be modelled by activating a special ic, similar to the ic_0 described above, to send messages to some of the index cells.

2. THE ACTIVE INDEX FOR THE SMART IMAGE SYSTEM

An active index is a dynamically changing net. As we shall see in Section 4, active index can be used to realize Petri nets, generalized Petri nets (G-nets), B-trees, etc. But its primary purpose is to serve as a dynamic index. We now illustrate by an example.

In current visual information systems, images don't have the capabilities to automatically respond to situational changes occurred in their environments. With advances in software and hardware technologies, images can play a more active role in their applications. For example, in the medical domain, after the examination of a patient's nuclear image, a doctor may want to compare images of the same patient at different states (exercising, normal, excited, etc.), then to examine images in the time domain (past histories), and finally to check images from other modalities. Instead of having the doctor to retrieve these relevant images with explicit queries and to convert and highlight the images properly, an active image can monitor the doctor's actions and provide the necessary information in proper formats on time.

To improve the effectiveness and efficiency of visual information systems, images should invoke actions by themselves. Depending upon applications, they can move themselves into proper local storage, pre-process themselves into the appropriate representations, and display themselves on the screen at the right time.

A smart image is an image with an associated knowledge structure, where knowledge includes attributes, routine procedures for how the image is used, and dynamic links to other objects for performing related actions.

A smart image knows what actions to take based on the user's interaction with the image and on the environmental changes to the images.

2.1 Smart Image Design for Large Image Databases

To illustrate how the active index can be applied to the Smart Image System, let us describe the 3-level active index for the Smart Image System, using the theoretical framework presented in Section 1.

2.1.1 Level-1 Index

The level-1 index is to pre-perform certain operations and specifically to prefetch image data. For each smart image, only one level-1 index cell can be activated. The input to this ic is the set of user messages of interest. A relevant user message regarding a smart image will be sent to this ic. This user message will (1) cause the ic to activate the appropriate level-2 ic, (2) post an output message to that level-2 ic, and (3) change state to the appropriate next state.

For example, if the ic is in s_{angio} and the user message indicates a Stenosis condition, then the ic will activate the level-2 $ic_{multi\text{-}modality}$, send an output message Image:Angio, Abnormality:Stenosis to $ic_{multi\text{-}modality}$, and change state to s_{muga}. The action of this ic in state s_{angio} is to prefetch all muga images of the patient.

The states in this level-1 index cell are the global states. Once the ic enters a global state, a selected group of the next level index cells can be activated. In the above example, the states correspond to the different image modalities, because when the user is viewing an image of a given modality, the index cell must be in that state, i.e. such states are observable.

State	Exam Name	Hotspot	Next State
1	START		
2	Angio	Ang.Hed.001	END
		Ang.Liv.001	Angio
		Angioma	END
		Arteriosclerosis	Angio
		Cicaterization	Angio
		Hemolysis	Angio
		Myoma	Nuclear
		Stenosis	Muga
3	Muga	Anaeurysm	Nuclear
		Angioma	Muga
		Anomaly	Angio
		Arteriosclerosis	Angio
		Hemolysis	Nuclear
		Hyperthropy	Nuclear
		Mug.Hrt.001	END
		Mug.Hrt.002	END
		Mug.Hrt.003	END
		Mug.Hrt.004	END
		Myoma	Nuclear
		Stenosis	END
4	Nuclear	Anomaly	END
		Myoma	END
		Nuc.Mug.001	END
		Nuc.Mug.002	Angio
		Nuc.Mug.003	END
		Stenosis	Angio
5	END		

Figure 1. State transitions for the level-1 index.

The state transitions for the level-1 index cell are given in Figure 1. As illustrated in Figure 1, from the current state, depending upon the user's input message (the condition), we can prefetch all relevant images of a given modality. Thus from State 2, if the condition is Stenosis, then we prefetch all the muga images of a specific patient and go to State 3. Figure 2 illustrates the relationships among images, hotspots and level-1 active index.

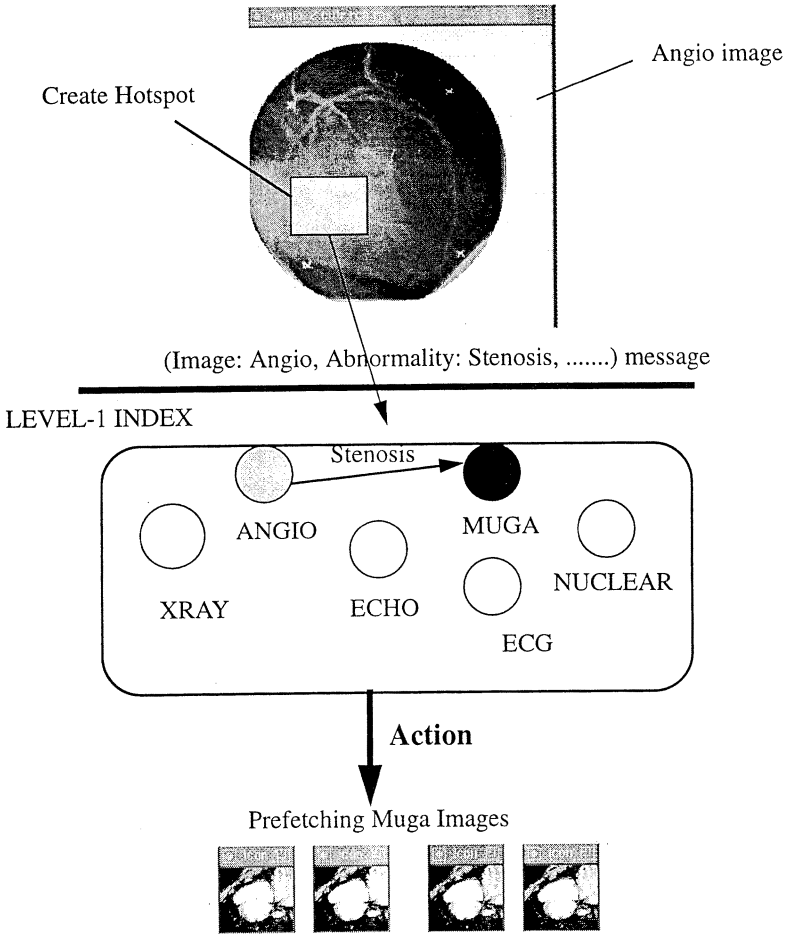

Figure 2. Relationships among images, hotspots and level-1 active index.

There may be too many muga images to be prefetched. Can we prefetch only a subset of these muga images? It depends on the following: (i) the filtering algorithm, (ii) the way images are organized in the class hierarchy, and (iii) the index cell construction algorithm.

Since the level-1 index cell is essentially a finite-state machine, there are effective learning algorithms to construct the cell from the past history of user messages. In principle, we can record every click made by the user as well as every text, voice or annotation messages. In practice, we use filters to extract the appropriate user messages and record them in the history. A moving window is kept, so that the recent history is used by the learning algorithm to construct the finite-state machine for the index cell. For example, the filtering algorithm may only extract user's identification of abnormality and accessing of image data: (Abnormality=Stenosis, Retrieve=Muga image taken on date-x), from the following history:

Doctor Name: Douglass A. Young
Patient Name: David Straker
SSN: 152-83-2745
SEX:M
Date of Birth: Sep 15 1953
Angio Image: Ang.Hrt.001, taken on Dec 19 1991, EID#=MHT-00010
CREATE_HS
Abnormality: Stenosis
RETRIEVE_IMAGE
Muga Image: Mug.Hrt.003, taken on Dec 10 1991, EID#=MHT-00030

In the smart image class hierarchy, images are divided into: (a) recent images (within one month), (b) fairly recent images (within one year), and (c) archival images (within ten years). The simplest index cell will just define a next state corresponding to ALL nuclear images. The more sophisticated index cell will have next states corresponding to (a) (b) and (c). The index cell construction algorithm will test whether date-x satisfies (a) (b) or (c) and then constructs the cell's next state(s). With this refined construction algorithm, only those images that are in (a), (b) or (c) will be prefetched.

2.1.2 Level-2 Index

The level-2 index is to perform hotspot-triggered actions in multi-modality study. If the user will make known to the system what study is being conducted, such as Coronary Artery Disease, Ventricular Function, and so on, the appropriate level-2 index cells will be activated based on the particular study.

A hotspot in a smart image, when triggered, may send messages to a level-2 index cell. For example, the input to an ic $ic_{left\_ventricular\_study}$ is the set of hotspot conditions such as abnormality, and quantitative data obtained from the image processing routine. An appropriate hotspot condition will

(1) cause the ic to activate another ic in level-2 or an ic in level-3, (2) post an output to that ic, and (3) change to dead state to de-activate itself.

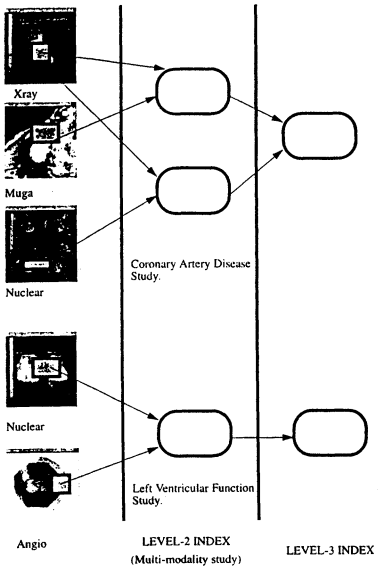

Figure 3. Level-2 active index.

Figure 3 illustrates the level-2 active index. In Figure 3, the active index is shown as a net of index cells. It should be emphasized that the arcs in this net are dynamic. Output arcs are specified when a live cell accepts and processes the input. They can change dynamically.

For example, the hotspot condition: LV_enlargement_abnormality, and heart volume quantitative data in nuclear image and the hotspot condition: Stenosis_abnormality, and low ejection fraction quantitative data in angio image, when triggered, will (1) cause the ic to activate the level-3 ic, (2) post the appropriate output message to the level-3 ic, and (3) de-activate $ic_{left\_ventricular\_study}$. Another hotspot may cause the ic to activate different output cells.

By using the technique of abstraction, we can combine the simpler cells into more complex cells with multiple input such as for multi-modality study. Similarly, some image processing functions may require multiple images as input. The detection of LV Enlargement and Stenosis in two different images may require two separate image processing functions. The two functions may be disjoint, and no image fusion is required. We will just test the logical predicates in the above example. On the other hand, for some cases image fusion will be required, and we must register the images, perform nonlinear transformations to correlate images, etc.

2.1.3 Level-3 Index

The level-3 index is to perform automatic linking and the retrieval of related and sometimes unanticipated information. When the user requests information (by clicking on some button), a message is sent to the ic. The input to an ic is therefore the set of retrieval requests. An appropriate retrieval request will (1) cause the ic to activate another (possibly remote) ic, (2) post an output message to that ic, and (3) change to initial state. The action is to send information to the original requester.

For example, the ic_{tumor}, with the initial input message tumor_found, may initiate a retrieval request, to retrieve all related information on that patient, and present it to the original requester (the physician who is interacting with the Smart Image System).

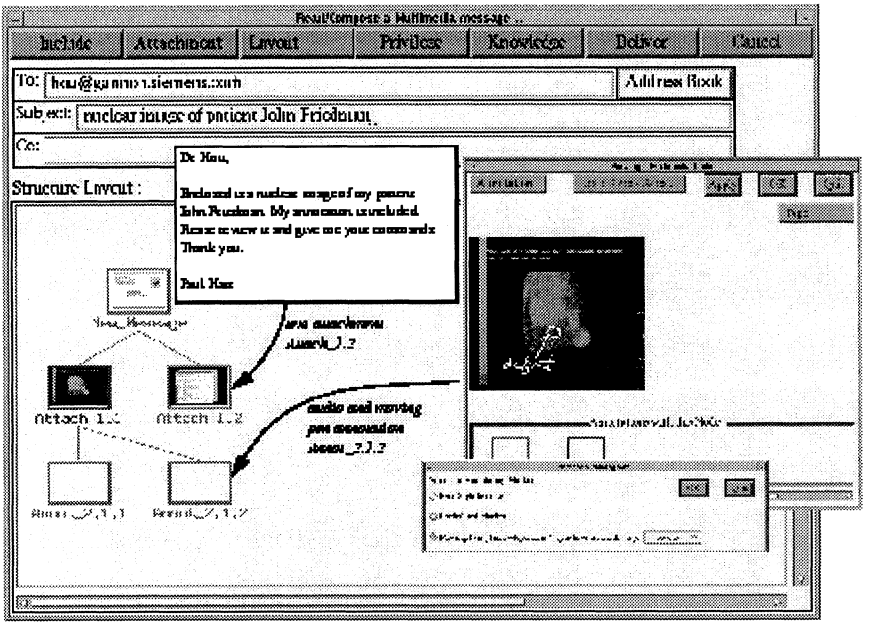

Figure 4. Annotation

Annotation as illustrated in Figure 4 could be considered unanticipated information. When an active index automatically performs linking and prefetching operations, unanticipated information can be included or not included. When the physician is making a decision, he needs the right amount of unanticipated information, but certainly he does not want every single new case in the medical journals. Thus, the active index with the appropriate action sequence determines what links to be established, and

what amount of information to be prefetched. Such flexibility makes the Smart Image System responsive to users' needs.

The function of the level-3 index is quite similar to that of the active index for information retrieval in hyperspace, which will be explained in the following section.

3. THE ACTIVE INDEX FOR INFORMATION RETRIEVAL IN HYPERSPACE

To retrieve information in the hyperspace, which is represented by a hyperstructure, we can associate an index cell with every recently accessed node in this hyperstructure. Thus the ICB corresponds to the set of all nodes in the hyperstructure, and IX a finite set of recently accessed nodes.

In a recent experiment, two months of tracing Mosaic usage in a university department show that about 40-45% of Mosaic files are accessed with high frequency. A relevant subset of the Mosaic objects transferred from remote servers are often used by other NCSA Mosaic clients. (In the university environment, the often looked-for information items are call-for-papers, books or technical announcements, new computer systems, etc.)

Therefore, to improve the system performance, frequently accessed information items should be prefetched and kept in the local cache.

The index cell can be constructed as follows:

It accepts a query q_k if $k > 0$, and activates the adjacent index cells, and posts q_{k-1} to them. The action performed is to prefetch information items satisfying the query.

A further refinement is to prefetch information items above a certain size. The justification is that we need only prefetch large information items, and small information items need not be prefetched.

As an example, if the original query is q_3, only cells within a distance of two links may be activated. Since we are posting a query to all the adjacent cells, if one of them accepts the query, the rest will no longer be able to process this query. Therefore, only three ic's on the following single path will be activated:

$$^{ic}q_3 - {}^{ic}q_2 - {}^{ic}q_1$$

If we post queries individually to the adjacent cells, the result is to activate all ic's where the distance from any ic to $^{ic}q_3$ is no more than two links. Consequently, more information items will be prefetched.

Both the viewer and the designer of a hyperstructure can add knowledge to the index cell as follows.

The viewer of the hyperstructure can send messages to the active index. For example, a clicking on a document indicates the invocation of a hyperlink. Certain index cell can then be activated.

We may also allow the viewer to add annotation to certain objects. In this case, the viewer can modify (a part of) g to activate the cell corresponding to the annotation object.

The designer of the hyperstructure can of course modify the g function to decide what new cells to activate. Thus, g contains the designer's knowledge. Therefore, the g function is central in capturing both the viewer's knowledge and the designer's knowledge.

The function g allows us to either add new cells to the system, or to stay with a predefined set of cells. We can set t_{max} to infinity, and exclude from g the dead state, so that index cells always remain live. We can further stipulate that g maps only to cells in IX. Thus the system can become a static index system.

On the other hand, if no query is posed, with finite t_{max} 's after a while all index cells will become dead. In other words, the active index is active, only as long as there are messages sent to the cells (or, to put it simply, only as long as there are users interested in certain information nodes of the hyperstructure).

From the above two examples, it can be seen that the major difference between an active index and a static index is that the active index is a dynamic collection of live index cells. The active index will change with time, as new index cells are activated and current index cells are de-activated.

4. THE COMPUTATION POWER OF THE ACTIVE INDEX

The active index is a powerful computing device. Its interconnections are dynamic, which makes it different from many other computing devices. By suitable restrictions, it can be used to realize Petri nets, the previously defined active index structure, conventional index structure such as the B-tree and a generalized Petri net called the G-net.

However, it is more general than all of the above, because in the active index the arcs are not fixed and static and may change dynamically.

4.1 It can realize the Petri net

Suppose the Petri Net is specified by (P,T,I,O) where P is the set of places, T is the set of transitions, I is the input function for the transition, and O is the output function for the transition.

We can construct an active index as follows: in the index cell base ICB, there are cells ${}^{ic}p_i$ corresponding to the places p_i , and cells ${}^{ic}t_j$ corresponding to the transitions t_j . The active index IX consists of ic_0 and these cells ${}^{ic}p_i$ and ${}^{ic}t_j$. Furthermore, they are perennial. The cell ic_0 is used to initialize the active index.

For each index cell ${}^{ic}p_i$ corresponding to the place p_i, ${}^{ic}p_i$ will accept any input x, because input can only come from transitions. After acceptance of input, the cell ${}^{ic}p_i$ activates the output index cells in ${}^{Tr}p_i$, where ${}^{ic}t_j$ is in ${}^{Tr}p_i$ if t_j is the output transition of p_i. The cell ${}^{ic}p_i$ then posts $({}^{Tr}p_i, {}^{x}p_i)$ to the output list.

For each index cell ${}^{ic}t_j$ corresponding to the transition t_j, it will accept the input set $\{ {}^{x}p_1, {}^{x}p_2,..., {}^{x}p_m \}$ where each p_i is the input place of transition t_j. After acceptance of input, the cell ${}^{ic}t_j$ activates the output index cells in ${}^{P1}t_j$, where ${}^{ic}p_i$ is in ${}^{P1}t_j$ if p_i is the output place of t_j. The cell ${}^{ic}t_j$ then posts $(\{{}^{ic}p_i\}, {}^{x}t_j)$ to the output list, for each ${}^{ic}p_i$ in ${}^{P1}t_j$.In other words, by Observation 1 of Section 1, the transition t_j can send messages individually to each output place p_i.

4.2 It can realize the previously defined active index structure

An active index structure can be defined to be a set of active index cells connected by arcs. Each cell has a number of input slots and output slots. Each input slot is connected to the output slot of another cell, and each output slot is connected to an input slot of another cell. The connected pair of input and output slots must have the same predicate. A cell R is enabled if tokens satisfying the input predicate flow into the cell. When the cell R is fired, one token each will flow to the input slot of another cell provided that the token satisfies the output predicate. When several input slots have identical predicates, they must all have tokens satisfying the predicate, before R is enabled.

The active index structure can be transformed to the equivalent Petri net where input slots with identical predicates are converted to input places for the same transitions, and output slots are converted to transitions leading to output places. But the current formulation of an active index is more natural and can be used directly to describe the originally conceived active index structure.

Basically, f({x_1, x_2, x_3},s) is 1, if x_1, x_2 and x_3 are inputs to the cell, and pred(x_1, x_2, x_3) is true. We can then post the output message to the output slot(s).

The current formulation is more general than the previously defined active index structure. The restriction of fixed input/out relationships has been removed. Index cells can be added/deleted dynamically, so that the active index varies in time.

4.3 It can realize conventional index structures

For example, the B-tree can be described by an active index. The technique is to provide different input to the index cell for the B-tree (called a B-cell). One input to B-cell indicates the insertion mode, and the other indicates the search mode. Other conventional index structures (index sequential, index direct, etc.) can also be described using similar techniques.

4.4 It can realize the G-Net

The active index system is conceptually derived from the G-Net system where each G-Net may invoke another G-Net. Therefore, each G-Net corresponds to an active index cell. When a G-Net is invoked, it accepts the input message. When it completes its computation, it sends messages back to the invoking G-net, and then de-activates itself. Furthermore, if we introduce the various levels of abstractions into G-net, we can also describe class hierarchies and other abstraction structures. This leads to methodological considerations in specifying the active index cell, the input message space X and output message space Y.

5. THE REVERSIBLE INDEX FOR FEATURE-BASED INDEXING

The level-2 active index shown in Figure 3 can be used for feature-based indexing. When a feature is detected in an image, a hotspot is triggered to send a message to an index cell, which in turn may send output messages to other cells.

Conversely, if we want to retrieve images having that feature, we need to reverse the flow in the index structure. In this section, we describe how to construct such a reversible index.

Suppose ic_i posts output message x_i to ic, $1 \leq i \leq m$. If we want to make ic accept these m messages as input, then $f(\{x_1, ..., x_m\}, 1) = 1$.We can tag every input message as (ic_i, x_i), so that the input message also indicates where it comes from. Thus we have the following modified f function,

$$f(\{(ic_1, x_1), ..., (ic_m, x_m)\}, 1) = 1.$$

Similarly, we can also tag the output message of this ic as follows, $g(\{(ic_1, x_1), ..., (ic_m, x_m)\}, 1) = (Ic', (ic,y), s', a)$ if one output message y is posted to all the ic' in Ic'; or $(Ic', \{(ic'_1, y / 1), ..., (ic'_n, y / n)\}, s', a)$ if the output y / j is posted individually to each ic'_j in Ic', $1 \leq j \leq n$.

For notational convenience, let $Ic = \{ ic_1,..., ic_m\}$, $V = \{ x_1,..., x_m\}$, $IcV = \{(ic_1, x_1),..., (ic_m, x_m)\}$,$Ic' = \{ ic'_1,...,ic'_n \}$, $W = \{ y_1,..., y_n\}$, and $Ic'W = \{(ic'_1, y_1),..., (ic'_n, y_n)\}$.

We can now describe the reversible index cell as follows.

Case 1: One output is posted to n output cells. For the original index cell, input $f(IcV,s) = 1$, and output $g(IcV,s) = (Ic, (ic,y), s', a)$. For the reversible index cell, we modify f and g as follows: input $f(\{(ic'_j, y)\}, r) = 1$ for $1 \leq j \leq n$, and output $g(\{(ic'_j, y)\}, r) = (Ic', IcV, r', b)$,where r, r' are new states corresponding to s, s', respectively. In other words, the reversed ic will accept y as the input, and posts x_i to ic_i individually as the output.

Case 2: An individual output for each output cell. For the original index cell, input $f(IcV,s) = 1$, and output $g(IcV,s) = (Ic', Ic'W, s', a)$. For the reversible index cell, we modify f and g as follows: input $f(Ic'W,r) = 1$, and output $g(Ic'W,r) = (Ic, IcV, r', b)$, where r, r' are new states corresponding to s, s', respectively. In other words, the reversed ic will accept $\{y_1 ,..., y_n\}$ as the input, and posts x_i to ic_i individually as the output.

In both cases, the action sequence b is left to be designed. If we first apply forward index to detect certain feature in an image, and then apply reverse index to find images having this feature, we can find images that are similar to the said image - similar in the sense of having the same features. Likewise, we can use forward index and then reverse index to find documents similar to a given document in the World-Wide-Web.

6. AN EXPERIMENTAL ACTIVE INDEX SYSTEM

The active index is a conceptual model. In actual implementation, the active index can be incorporated into almost any application system. For the Smart Image System, for example, the hotspot lends itself to a natural coupling with the active index, in the sense that once a hotspot is triggered, a

message is posted and the corresponding index cell is activated. For the Mosaic application, the clicking on a hotword has similar effects.

We have built an experimental active index system. The heart of the active index system is the IC_Manager, which performs the functions of receiving incoming messages, activating index cells, performing actions, and handling outgoing messages.

As illustrated in Figure 5, although in theory ic_1 can directly send message m_1 to ic_2, and ic_2 can directly send message m_2 to ic_3 residing in another machine, in practice every message must go through the IC_Manager.

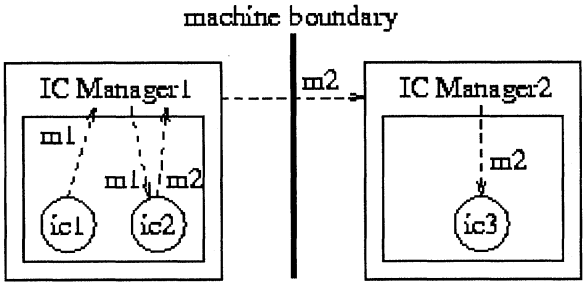

Figure 5. The IC Manager.

Another implementation approach is to realize each cell as a separate process, but that will result in costly interprocess communication overheads. Since efficiency is a major concern, that approach was not adopted.

The core of the IC_Manager is described as follows:

```
IC_Manager(message)
begin
 if message contains ic_id
   begin  /*the message is for a specific ic that should already exist*/
     locate ic_id in IX;
     add message to input_list; end ;
 if message contains ic_type
   begin  /*the message is for an ic to be created*/
     locate ic_type in ICB;
     create a new ic_id;
     add a new ic instance to IX;
     add message to input_list of this ic;
     add ic_id to the output_list of the output ic; end ;
 while there is next ic_id in IX
```

```
    begin check whether message should be accepted;
     if message should be accepted
       if message has not been accepted by another ic
        begin  accept this message and remove it from output_list;
          process this ic; end
     end
  end
```

In theory, the index cell base ICB can be infinite. In practice, it is necessary to maintain a library ICB of a fairly small number of generic ic's, so that the user can create customized cells with ease.

For the Smart Image System, it is also necessary to have a separate collection of generic index cells for each level of the three-level index.

The ICB and IX are implemented as linked lists of C structures. Whenever there is a request to activate (or create) a new index cell, a new cell is obtained from an available list space. Conversely, a dead cell is returned to the available list space.

The IC_Manager has a domain-independent part and a domain-specific part. The domain-specific part contains the specific routines used by the ic's to perform predefined actions. It also identifies and structures the external messages to be sent to the IC_Manager.

This clean separation of domain-independent and domain-specific parts makes it easy to adapt the IC_Manager to a new application system.

The IC_Manager is written in standard C codes and can easily be compiled together with the intended application system, on workstations as well as PCs, to produce a customized application system with built-in active index.

Another important tool is the IC_Builder, which is a visual user interface enabling the designer to visually design new index cells from scratch, or customize an ic based upon a generic ic from ICB. This tool was introduced in Chapter 4 and will be described in detail in Chapter 8.

For a WWW client such as the Mosaic, we can invoke the active index from Mosaic, so that the user's clicks generate retrieval requests. For information retrieval in hyperspace, the simplest approach is to use only level-3 index cells to link and retrieve information. Figure 6 illustrates the experimental Mosaic-IC at work, where the background window on the right displays the trace of instantaneous descriptions of the active index, and the action_icons in the upper-right corner show the actions performed.

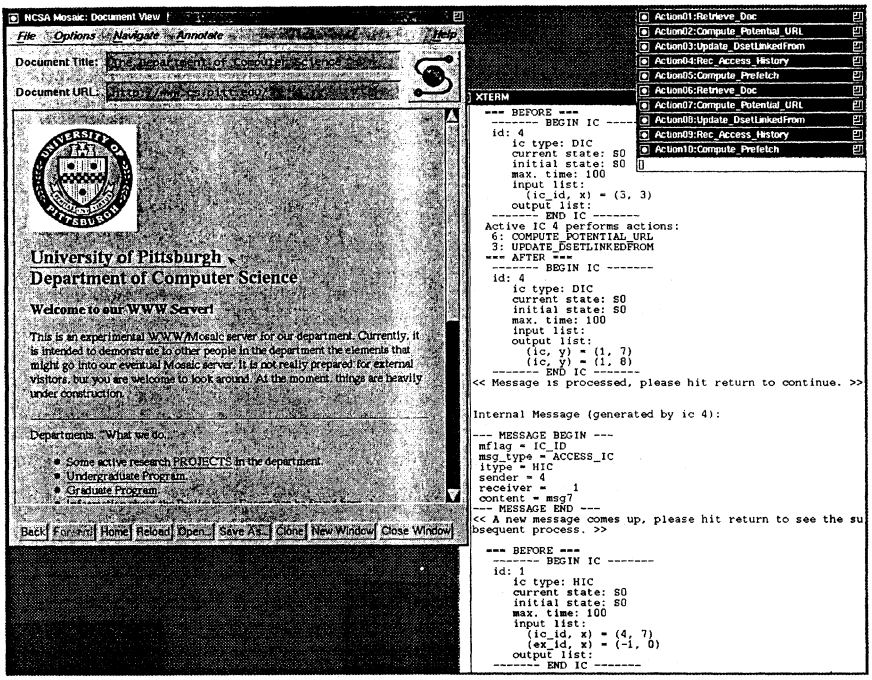

Figure 6. The experimental Mosaic-IC system.

For further research, the user can be modeled using level-1 index, the information abstracted using level-2 index, and information items linked and selectively presented using level-3 index. Moreover, using the reversible index, we can find documents similar to a given document. Special generic cells can be designed, to do range-based retrieval and incremental knowledge acquisition.

In implementing the experimental active index system, we decided to provide each cell with an internal memory. Theoretically, the internal memory and the state together define the true state of the cell. In practice, it is more convenient to have an internal working memory, so that the cells can cope with different situations flexibly. The internal memory is a C structure, so that the user can include special routines in the domain-specific part of the IC_Manager to manipulate it.

Using the experimental active index system, we quickly produced customized Smart Image System (SIS-IC), Mosaic (Mosaic-IC), B-Tree (BT-IC) and Medical Personal Digital Assistant (MPDA-IC). Thus the experimental active index system serves as a prototyping tool to enhance application systems with active indexes.

7. DISCUSSION

The active index introduced in this chapter possesses the following desirable characteristics: (a) The active index can be used to initiate actions and is active rather than passive. (b) Only a few index cells are activated as needed, so the index is partial rather than total. (c) The index is dynamic and can evolve, grow and shrink. (d) The index cell can send messages to the user in its action sequence and therefore the index can become visible to the user. (e) Finally, with the reversible index, the active index can be used to process imprecise queries and perform similarity retrieval.

The following topics require further research: (a) The index cell reversal technique described in Section 5 enables us to extract features from an image to construct feature-based index cells, and then retrieve images containing such features using these index cells. The feature-based index cells, like other index cells, have a finite lifetime. If they don't receive any messages for a while, they die. Dead index cells can either be eliminated, or archived to tertiary storage. Therefore, the system will not be burdened with excessively large indexes. The algorithms for index cell reversal need to be carefully designed, so that we can perform feature extraction and feature-based indexing using the same index structure.

(b) The time bound t_{max} limits the size of the active index, so that it will not grow too large. Inactive cells of the active index will be removed or archived automatically. A research issue is to study the stability of time-varying active indexes. Under what conditions will an active index become dead? Moreover, how can we adjust t_{max} so that the index is always below the storage constraint?

(c) The knowledge is contained in the two functions f and g. The function f restricts the inputs to be processed. The function g specifies the output, what cells to activate, next state and the action sequence. When, for example, the designer or the viewer of a hyperstructure wants to add knowledge to the cells, we need algorithms to allow incremental addition of knowledge by systematically modifying the g function.

Chapter 6

Semantics: Teleaction Objects

Teleaction Objects (TAOs) possess private knowledge specific to the object instances. The user can create and modify the private knowledge of a Teleaction Object, so that the Teleaction Object will automatically react to certain events to pre-perform operations for generating timely response, improving operational efficiency and maintaining consistency. Moreover, Teleaction Objects also possess a hypergraph structure leading to the effective presentation and efficient communication of multimedia information. The Active Multimedia System (AMS) is designed to manage the Teleaction Objects. In the AMS, the private knowledge of each Teleaction Object is realized by the index cells of Active Index. The applications to smart multimedia mail and multimedia information retrieval are described to illustrate the usefulness of Teleaction Objects.

1. INTRODUCTION

We describe the Teleaction Object (TAO) which is a multimedia object with associated hypergraph structure and knowledge structure. Recently distributed multimedia systems have become a common requirement for more and more applications [Berra90]. Although different media types have different characteristics in the size, resolution, storage method, transmission method, compression algorithm, presentation technique, etc., two central problems are common regardless of the application: presenting multimedia information effectively and transmitting multimedia information efficiently. It is therefore desirable to have a common approach for multimedia data modelling, which can lead to the solution of both problems [Little90a] [Znati93]. Teleaction Object, with its rich classifications of node types, media

types, and link types in the hypergraph structure, enables us to design algorithms to decide the priority in both communication and presentation for multimedia information according to the current environment and restrictions. With the different levels of knowledge specified by the system and the end-user, the TAO can react automatically to certain events.

An Active Multimedia System (AMS) is designed based upon the concept of TAOs [ChangH95b]. The AMS provides mechanisms for manipulating the TAOs so that the system can gather the private knowledge from a TAO instance and merge it into the knowledge base. The AMS provides mechanisms for maintaining the knowledge so that the TAO can automatically react to certain events to pre-perform operations for generating timely response, improving operational efficiency and maintaining consistency. The AMS also provides the tools so that the user can create his/her own TAO, and implement his/her own applications to handle the TAO. The Teleaction Object is a conceptual model. In actual implementation, the Teleaction Object can be implemented as objects in an object-oriented system. For a Multimedia Mail System, for example, a mail can be regarded as a Teleaction Object.

The approach of Teleaction Object model can support and improve many applications. For example, in the global information system such as the World-Wide Web (WWW) [Berners-Lee92], the user can navigate in the hyperspace supported by the network. But the major limitation is that the information-requesting mechanism constrains any transaction between the user and system to be passive. The Teleaction Object approach can be incorporated with the WWW browser such as the Mosaic. Therefore, the knowledge part will support the active actions in network information systems, such as two-way interaction [Dimit94], or pre-fetching.

Another application domain for Teleaction Object is delayed conferencing [Hou94]. The objective for delayed conferencing is to allow a group of participants to exchange information, including existing and newly created information, in a timely and consistent manner. Such delayed conferencing is based on the multimedia-based message delivery mechanism and exchanged information consisting of multimedia objects. Current multimedia mail systems plus groupware systems [Borenstin92] [Goldberg92] may support delayed conferencing. However, the limitation is that the system only executes/takes actions when the message is read by the recipient, or when the recipient answers certain questions. There are occasions when it is required to take action even before a message is read, or the recipient's environment is changed. The user should be allowed to define the event and the condition for actions to take place. The Teleaction Object approach can be applied to delayed conferencing because it supports

multimedia information exchange and maintains the knowledge at different levels to automatically react to the events.

This chapter is structured as follows. The next section gives the definitions and the motivation of the definitions for a Teleaction Object with both hypergraph structure and knowledge structure. Section 3 presents a scenario to explain how the Teleaction Objects work in the Active Multimedia System. The system architecture of the Active Multimedia System is described in Section 4. Based on the Active Multimedia System and the Teleaction Object approach, a working Smart Multimedia Mail application is illustrated in Section 5. Section 6 discusses application to multimedia information retrieval where an index cells hierarchy leads to user-specific pre-fetching of multimedia information. Section 7 discusses some of the contributions and identifies ongoing and future research.

2. DEFINITION OF TELEACTION OBJECTS

A Teleaction Object can be as simple as a single piece of information without connection or relation to any other objects. Or we can combine several TAOs in certain connections into a new complex TAO and/or add certain knowledge to a TAO. Basically, the TAO is further refined as two parts (G, K): hypergraph G and knowledge K. For a TAO, the hypergraph G is used to describe the connections and relations between the sub-TAOs within it. The knowledge K is used to describe the actions.

The Active Multimedia System AMS is a system that manipulates and maintains the Teleaction Objects and also provides the basic tools and methodology such that users can implement their own applications to handle the TAOs. The AMS is similar to the operating system of a computer, but instead of allocating and maintaining the resources AMS allocates and maintains TAOs. Section 4 will give the details of AMS architecture.

2.1 Definition of TAO Hypergraph Part

The hypergraph structure G plays an important part in TAO. Basically G is a graph structure (N, L), where N is a set of nodes and L is a set of links. Each node in G represents another Teleaction Object (TAO) and a link represents a specified relation or connection between these TAOs. By further refining the types of nodes, media and links, we can utilize this hypergraph structure for the dual purposes of regulating multimedia communication and generating multimedia presentations.

(a) Node types

There are essentially two types of nodes in the hypergraph structure G: basic node and composite node.

A *basic node* is defined as a terminal node in the hypergraph structure. The real media data is associated with the basic node and each basic node contains one and only one media data type. In other words, the basic node is the smallest unit of a TAO. For example, an image is a basic node. The media types for the basic node are defined below in (b).

A *composite node* is defined as a non-terminal node in the hypergraph structure. It contains a number of basic nodes and/or composite nodes. The composite node is a group of data instead of a single real media data. For example, a book chapter is a composite node in the hypergraph structure of a book because it contains sections, pictures and tables. By using the composite nodes, we not only can present the hierarchy of the multimedia object, but also can apply knowledge to a group of multimedia objects.

(b) Media types

Each media type has different characteristics of size, cost, user interface, storage hardware, interpretation, etc. Since applications choose different media types under different situations and considerations, knowing the media type is helpful for the system to optimize its performance. In our system, we define media types for the basic node in TAO as: text, graphics, image, moving-graphics, moving-image, audio, video, form and live-demo; while composition is for a composite node.

- *text* is coded alphanumeric data. It is the most basic media type for most multimedia applications.
- *graphics* is formatted picture data.
- *image* is pixel formatted picture data.
- *moving-graphics,* also called animation, is the formatted data of a graphics sequence.
- *moving-image* is a sequence of image frames.
- *audio* is formatted sound data.
- *video* is a combination of synchronized moving-image and audio.
- *form* restricts user input, possibly with additional formula to generate the content automatically.
- *live-demo* is a program that can be run to provide an interactive demo.
- *composition* is the media type of a composite node in TAO.

(c) Link types

We represent the specific relations and connections between nodes by using different types of links: attachment link, annotation link, reference link, location link and synchronization link.

An *attachment link* links a composite node A and another node B which is either a basic node or a composite node. It indicates that node B is a component of node A. This is the essential relationship between nodes in a multimedia hypergraph structure.

An *annotation link* links node A with node B, and both nodes can be basic node or composite node. This type of link specifies node B is an annotation associated with node A. The annotation has a different meaning from that of the attachment because the annotating node is an explanation or a synopsis of the annotated object, but it is not a necessary component required to construct the annotated node.

A *reference link* specifies there should be a navigation path from one node A to another node B when the user is browsing in the hypergraph structure G. Node B should be a node with no bundled node (to be explained later) as its ancestor.

A *location link* specifies the spatial relationship between nodes for their presentation. The number of nodes linked by a location link can be more than two and the type of each node can be basic node or composite node. We have developed a fuzzy relation language FRL [ChangH95a] to describe the spatial relationship in both horizontal and vertical directions. For example, image A and image B are located side by side and touching one another in the horizontal direction, which can be expressed by the location link with the relation language such as "X: A] == [B".

A *synchronization link* specifies the temporal relationship between nodes for their presentation. The number of nodes linked by a synchronization link can be more than two and the type of each node can be basic node or composite node. Similar to location links, we can use the same relation language to describe the temporal relationship for the synchronization links [ChangH95a]. For example, 5 seconds after image A is displayed we would like to play a moving-graphics B and an audio C at the same time, which can be expressed by the synchronization link with the relation language such as "T: A] < 5 [B; T: [B == [C".

For the purpose of facilitating multimedia presentation, multimedia communication, hyperlinking, and replacement, there is a special feature for nodes known as bundled nodes. A *bundled node* can be either a basic node or a composite node. A bundled composite-node serves as a single unit with all its components bundled together. For a bundled basic-node, its sole purpose is presentation. Using the bundling concept greatly simplifies the

specification of multimedia presentation. For instance, if a basic node containing audio is a bundled node, it can only be played from the beginning to the end, and it cannot be played half way. For another instance, if a composite node containing three basic components (text, image, and audio) is a bundled node, we cannot just present anyone of its components without presenting the other two.

A function a_bundle(N) is defined for node N as follows:

a_bundle(N) = nearest bundled ancestor of node N (excluding itself) in G when traversed along attachment or annotation links from N
= NIL if such node does not exist.

There are two constraints for bundled nodes. The first is regarding the relation links which are either location link or synchronization link: a relation link can be established between two nodes N and M only if (1) a_bundle(N) is the same as a_bundle(M); or (2) one of them is a_bundle of the other. The second constraint is regarding the reference links: a reference link should only link to a node N such that a_bundle(N) = NIL.

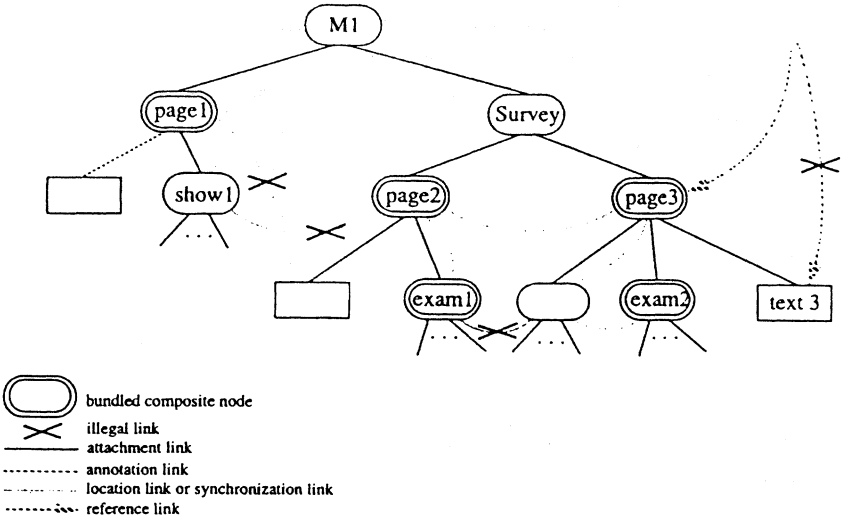

Figure 1. Example of legal and illegal links in a hyperstructure.

Bundled nodes have three purposes. First, restricting relation links at some nodes will retain only the meaningful relations expressed by Fuzzy Relation Language (FRL) Briefly speaking, FRL is used to express

temporal/spatial relations in presentation between nodes. Without bundled nodes to limit the use of relation links, many cases would occur which do not make sense. As explained in Figure 1, the double-edged nodes are bundled nodes, the lighter arcs represent the relation links, and the crosses indicate where the relation/reference links are illegal.

Second, restricting reference links at some nodes will maintain the integrity of presentation. As shown in Figure 1, the dotted arrow linking to the basic node `text3' is illegal because `text3' is one component of the bundled node `page3'. The presentation system cannot display `text3' without displaying all other components of `page3'. Thus, the reference link to `text3' should be moved up to `page3'.

Third, replacement of a bundled node by another representation-equivalent bundled node becomes possible. We can devise algorithms to replace a bundled node by another bundled node, as long as the presentation effects are comparable by some measurements. For example, the animation of a demo can be replaced by a live demo (actually running the program). Replacing a video by a still image with dubbed sound track is another possible replacement.

2.2 Motivation for TAO Hypergraph Part

In a distributed environment, communication and presentation of multimedia objects is both time consuming and space consuming. We need to transfer and display multimedia objects efficiently, effectively, and properly. For example, when the network traffic is high, we may send the main content of a multimedia object first, transmitting important images in low resolution, and leaving out the less important parts for later transmission. When we present multimedia objects, we need to know the order of presentation and possibly pre-fetch other data. In other words, we need to determine the priority of the transmission sequence and the presentation order for multimedia objects. Thus, a rich set of node types, media types and link types will enable us to use the hypergraph structure G to control communication and organize presentation. An example is given in Section 3. The different types of nodes and their structure in G provide useful information of different relationships between the TAOs. It is feasible to design algorithms to traverse the hypergraph G and determine the transmission sequence and the presentation order according to the currently available communication bandwidth, recipient's environment, link types, node types, media types and structure in G [Lin94].

Given a multimedia hypergraph G, we can apply the following algorithm to generate its Multimedia Data Schema (MDS) [Lin94], which controls the

synchronization between time-related data streams. The MDS is similar to the Object Composition Petri Net (OCPN) [Berra90], [Little90a].

1. Let g be the subgraph of the hypergraph G so that nodes in g can be traversed from the root of G via 'attachment' and 'annotation' links only.
2. For each node n in g, let level(n) be the level of n in the breadth-first spanning tree of g.
3. In g, for each synchronization link, which connects two nodes, say Mi and Mj:
   3.1. Let Mk defined as Nearest-Common-Ancestor(Mi, Mj), and let
   3.2. k = level(Mk).
   3.2. /* CASE 1: Ancestor-Descender relationship */
   If level(Mi) ≠ level(Mj) and Mk ∈ {Mi, Mj} /* and assume Mi = Mk */ then
   3.2.1. Let Mj' = Ancestor(Mj) at level k+1.
   3.2.2. Propagate temporal information from Mj up to Mj'; and create a new synchronization relation between Mi and Mj'.
   3.3. /* CASE 2: Cousin-Cousin relationship or */
   /* CASE 3: Distant relationship */
   If (level(Mi) = level(Mj) and level(Mk) + 1 ≠ level(Mj)) or (level(Mi) ≠ level(Mj) and Mk ∈ {Mi, Mj}) then
   3.3.1. Let Mi' = Ancestor(Mi) at level k+1; and
   Let Mj' = Ancestor(Mj) at level k+1
   3.3.2. Propagate temporal information from Mi up to Mi' and
   3.3.3. from Mj up to Mj'; and create a new synchronization relation between Mi' and Mj'.
4. Recursively generate MDS by using Algorithm described in [Lin94].

The transitions in MDS indicate points of synchronization and the places represent the media presentation processing. In other words, we should present the multimedia objects in a specified order according to the MDS. Again there are several considerations to generate an effective presentation sequence, such as the location and synchronization link relation between the TAOs, computer hardware, capability to display different media type, storage size etc. An example for MDS is given in Figure 5 of Section 3.

Given a Multimedia Data Schema (MDS), there is an algorithm to generate its Multimedia Communication Schema (MCS) [Lin94] for an efficient transmission sequence. There are several considerations to generate an efficient transmission sequence, such as the size of the TAOs, the type of the TAOs, the structure of the TAOs, the link relation between the TAOs, computer hardware, the capability to display different media type, the

storage size, bandwidth of communication, etc. For example, if we adopt the progressive transmission technique in the communication schema, the low-resolution image or the low-quality audio will be sent first if the network traffic is high. Then we try to transmit the higher quality data if possible. For those nodes connected by a synchronization link, the multimedia objects will be transmitted according to the synchronization requirements. Also, the nodes linked by an attachment link should have higher priority than the nodes linked by the annotation link and reference link because the user may want to see the content of top-level TAO first. Or the nodes closer to the currently viewed node should have higher priority than the nodes farther away from the currently viewed node in the transmission sequence.

Moreover, given a hypergraph G, we can design an algorithm to decide the pre-fetch sequence in order to improve the efficiency of multimedia browsing. Thus, only needed data become available based on the currently viewed TAOs. The pre-fetch sequence is changed dynamically and performed in the background and does not require monitoring by the user.

From the above discussion, it can be seen that the sets of node types, link types, and media types will provide the information needed for: (a) automatic scheduling of synchronization in communication; (b) automatic generation of proper presentation for specified spatial and temporal relations and (c) automatic pre-fetching of potential multimedia objects.

2.3 Definition of TAO Knowledge Part

Without specifying the part of knowledge K in a Teleaction Object, a TAO is just another hyper-media object. In fact, the simplest media object is a TAO whose hypergraph part is reduced to a simple node and whose knowledge part is empty. We can classify the knowledge of TAO into four levels:

System knowledge is associated with all TAO instances. It handles all generic operations that are applicable to all TAOs. This knowledge defines the default intent for each TAO. For example, checking the privilege for viewing each TAO; or keeping the history log; or pre-fetching other TAOs from remote servers when these TAOs are within a certain distance from the TAO currently being viewed in the hypergraph G. This knowledge is created by the system.

Environment knowledge is associated with all TAO instances belonging to a special user. A user may want to customize his/her AMS operations. So the user can add or remove some knowledge to his/her local knowledge base. For example, user A might add a new knowledge to purge TAOs whenever the system storage is low. The user can also overwrite some system knowledge, for example, he/she can change the distance criterion for pre-

fetch. The environment knowledge can be generated by either the system or the user.

Template knowledge is associated with a group of TAO instances in a predefined format. For frequently used TAO formats, a hypergraph as well as the associated knowledge can be provided, such as the time scheduler, the weekly work report generator, the resume, etc. It can be generated by either the system or the user.

Private knowledge is associated with one special TAO instance. This is the most important knowledge for the user because it carries individualized knowledge, which the user has created for the single TAO instance. It is generated by the user.

There is a local knowledge base for each user. When a user registers for the first time, AMS will create a local knowledge base for this user and initialize it with the system knowledge. After that, the environment knowledge is merged, withdrawn or overwritten into the local knowledge base. As time goes by, lots of template knowledge and private knowledge will be merged into or removed from local knowledge base when the TAOs become alive/dead. Therefore, the knowledge base is local because after the initialization, the inclusion of different environment knowledge, template knowledge and private knowledge leads to different user's local knowledge base.

For a Teleaction Object TAO instance, as discussed in previous sections, it is represented by the (G, K) pair. The knowledge part K of a TAO instance could be either the template knowledge if the TAO is a predefined template instance, or the private knowledge if the TAO is an individual TAO instance. When the Teleaction Object (G, K) becomes available, K is merged into the user's local knowledge base. Similarly, when the TAO becomes unavailable, the corresponding K should be withdrawn from the user's knowledge base. When a TAO moves from one user A's local knowledge base to another user B's local knowledge base, K will be copied into B's local knowledge base. With a better algorithm, part of the K will be merged into A's local knowledge base and part of K will move along with the TAO and be merged into B's local knowledge base.

Conceptually, there is a priority for overriding the existing knowledge as follows: private knowledge, template knowledge, environment knowledge, and system knowledge, where the private knowledge has the highest priority and the system knowledge has the lowest priority.

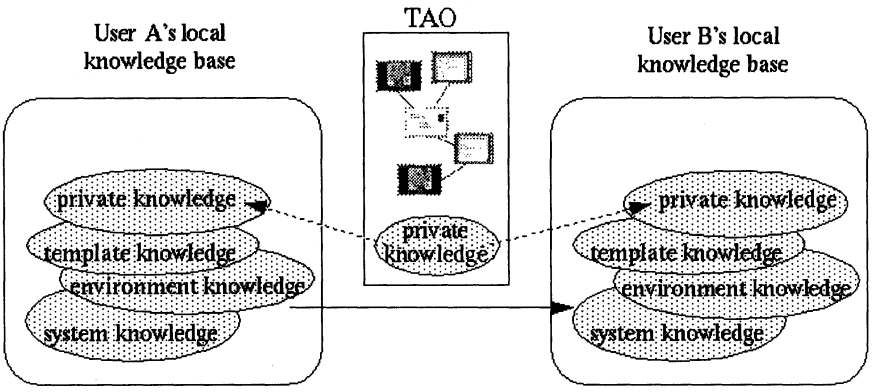

Figure 2. User A sends a Teleaction Object to user B.

As illustrated in Figure 2, the solid line indicates the transmission of the TAO from user A to user B, and the dotted lines indicate the sharing of private knowledge of TAO in both system A and system B. Private knowledge has the highest priority to override others in the local knowledge base.

From the object-oriented point of view, each TAO instance and each knowledge piece has its own class name. The hierarchy class name is indicated in the hierarchy, such as "root.TAO.mail". Therefore, once a TAO instance is available in the AMS system, knowledge with the same class name and super-class name will be attached to it. If the knowledge pieces have naming conflicts, the system will use the one with the longest class name inheriting and overriding knowledge at different levels. For example, once a TAO instance with the class name "root.TAO.mail" is available in the AMS, the system will attach the knowledge with class name "root", "root.TAO", and "root.TAO.mail" to it. And if two knowledge pieces have the same name, "pre-fetching", but with different class names, one being "root.TAO" and another being the class name "root.TAO.mail", the system will attach the one with "root.TAO.mail". In other words, it overrides part of the inherited knowledge.

In our model, the system knowledge in AMS is implemented with class name "root" and/or "root.TAO". All the instances of TAO will be attached to it since each TAO instance has a class name beginning with "root.TAO". Environment knowledge has the ability to modify/add the knowledge piece with class name "root" and/or "root.TAO", such that the user can customize his/her own local knowledge base. Template knowledge is used to build up new knowledge pieces with a nesting class name, such as "root.TAO.mail" or "root.TAO.mail.resume". Therefore all TAOs with the same class name in

depth, will be attached with this knowledge piece. Finally the private knowledge is attached to only one single TAO instance. We can implement the private knowledge piece with the class name ended with the identical TAO Id number, e.g. "root.TAO.mail.resume.1234". That means only this single TAO instance belongs to this class. Therefore, the private knowledge is attached to this TAO instance only.

2.4 Motivation for TAO Knowledge Part

Although we can abstract some useful information from different types of nodes, media and links of the TAO to improve the communication and presentation for multimedia data, without the knowledge part the TAO is just a complex hypermedia object to be used in a passive way. In order to change the TAO from being passive to active, we add the knowledge part for a TAO. Therefore, once a TAO becomes available in the AMS, it will add its own knowledge to the local knowledge base in AMS. And whenever a specified event occurs, related actions take place according to the knowledge. In other words, TAO is an active object.

By using the concept of system knowledge level and environment knowledge level, we can accomplish a customized system for individual users. After user's AMS is initialized by the system knowledge, it can be customized by overriding the system knowledge with environment knowledge. Since both knowledge levels are associated with class name "root" and/or "root.TAO", these knowledge pieces are applied to all available TAO instances in the AMS. Basically, these knowledge pieces are more related to the environment changing, pre-fetching, or pre-processing.

In certain cases, only the user has the best knowledge on how to manage the object. Therefore, we allow users to specify the knowledge within one TAO, which means some special knowledge is associated only with this TAO but not with any other TAO. This is the object's private knowledge. On the other hand, some TAOs are expected to share the template knowledge. Both template knowledge level and private knowledge level are for a special group of TAOs, the only difference is that a single TAO is in the groups of private knowledge. Basically, the template knowledge is more related to the application-specific features, such as mail or resume, while the private knowledge is related to really personal matter for a special single TAO.

From the above discussion, using knowledge in the TAO model will allow the objects to become active, and using different knowledge levels will allow the user to customize AMS and to specify private information. These features enable the Teleaction Object to become an intelligent active object, and the AMS to be more flexible.

3. A SCENARIO

Let us now present a scenario as an example. A project manager Smith is preparing a proposal for his boss Kessler and his group members Wang and Larson. His proposal M1 contains several pages of text, audio and images and also links to a confidential report M2 for his boss Kessler only. Two annotations about the image are also included in his proposal. The hypergraph structure of the proposal is shown in Figure 3.

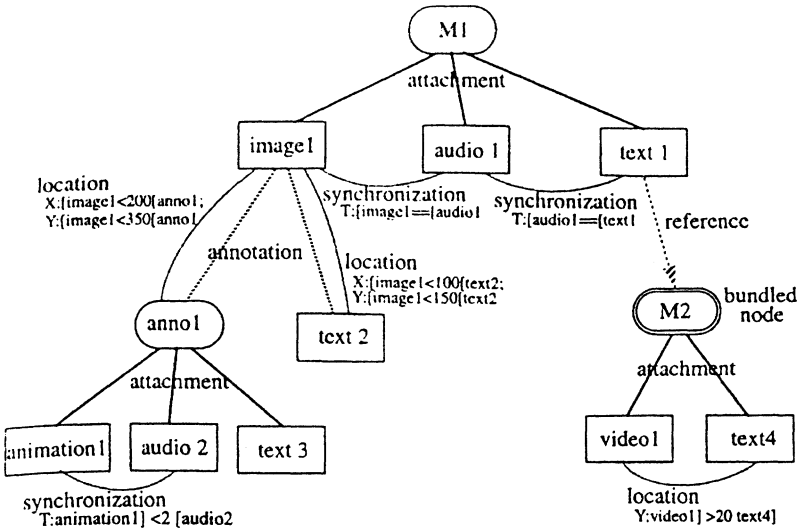

Figure 3. The hypergraph structure of the proposal.

In Figure 3, the rectangle denotes a basic node and the rounded rectangle denotes a composite node. The attachment links indicate the composition of the composite nodes. For example, the proposal M1 is composed of text, audio and image data; the confidential report M2 is a bundled node composed of text and video data; and the annotating object "anno1" is composed of text, audio and animation data. Two annotation links in this example specify that the image has two annotations on it; one annotation is a basic node for text data and the other annotation is a composite node. The reference link indicates there is a navigation path from text of M1 to the bundled confidential report M2 (in this case, for boss Kessler's eyes only). Location links specify where the annotations, anno1 and text2, are with respect to image1 and specify the layout of the confidential report M2. In the former case, the location links are between parent nodes and child nodes, while in the latter case, the location link is between sibling nodes and it

specifies that video1 is 20 units above text4. The synchronization link specifies the temporal ordering in the presentation. In this example, the synchronization link between animation1 and audio2 specifies that the audio2 should be delayed 2 seconds after having finished playing the animation1 while the two synchronization links between image1, audio1 and text specify they should be started simultaneously for M1.

According to the hypergraph structure, we develop the Multimedia Data Schema, shown in Figure 4, when any one of the group members decides to browse the proposal M1. Token flow in a Multimedia Data Schema illustrates the presentation of multimedia objects, while as token flow in a Multimedia Communication Schema illustrates the transmission of multimedia objects.

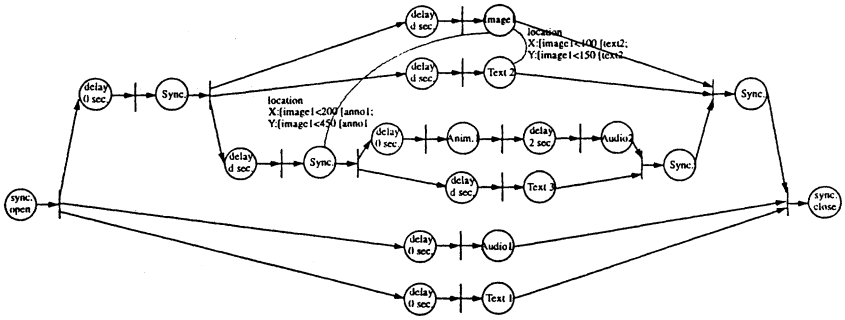

Figure 4. Multimedia Data Schema.

After Smith creates a Teleaction Object M1 with the hypergraph structure shown in Figure 3, he may want to accomplish several tasks just for this special proposal M1 only. The tasks are: (pic.a) the boss Kessler and the two group members should read this proposal in two days and send back their responses; (pic.b) the responses from Kessler and at least one of the two group members are needed in order to proceed to the next step in proposal preparation; and (pic.c) this proposal should become obsolete after one week. In this case, Smith will include the private knowledge with the TAO.

In our example, there are four local knowledge bases for the four users: Smith, Kessler, Wang and Larson, respectively. When these four users first join the AMS, all have the same local knowledge bases that are initialized with the system knowledge. For example it may include the following system knowledge: (sic.a) If the user has read part of a multimedia object, pre-fetch all the information within two hypergraph links from the object being viewed, in the associated hypergraph structure; (sic.b) If the user intends to read a part of the multimedia object, check whether the user has

the privilege or not; (sic.c) If any annotation part is added, do the version control and history checking; and (sic.d) If any multimedia object is obsolete, remove it.

After each user has the initial local knowledge base, the three other kinds of knowledge will be merged into, overwritten on, or removed from the local knowledge base. For example, the following environment knowledge is added for Smith: (eic.a) If any new TAO contains the keyword "FYI", make a copy to folder "FYI-1994"; and (eic.b) If the user has read part of a multimedia object, pre-fetch all the information within three hypergraph links from the object being viewed, in the associated hypergraph structure (perhaps because Smith has a larger storage allocation in his system).

The environment knowledge eic.a is merged into the user Smith's local knowledge base, while the environment knowledge eic.b overwrites the system knowledge sic.a. Therefore, all the TAO instances available in Smith's local system are different from that of the other three users because when a new "FYI" TAO becomes available, Smith's local knowledge base will have a backup copy in a folder while other's local knowledge base will not; and TAOs are pre-fetched within three links distance instead of two links distance in Smith's local system.

Returning to Smith's proposal M1, he creates the private knowledge for M1 and sends it to the other three users. Part of the private knowledge may go with the message while some knowledge may still be kept in the knowledge creator's local knowledge base. In our example, the two private knowledge, pic.a and pic.b, will be sent along with the M1 to the recipient's local knowledge base. In other words, these two private knowledge pieces will be merged into Kessler's, Wang's, and Larson's local knowledge bases. As for the last private knowledge pic.c of proposal M1, it is not sent along with M1, instead it is merged into Smith's local knowledge base. The concept is also illustrated in Figure 3.

Also in this example, we can expect that the private knowledge pic.c will override the system knowledge sic.d. If this proposal M1 is expired, all information of M1 is stored instead of being removed.

4. THE AMS ARCHITECTURE

We have discussed the two important components of a TAO: the hypergraph structure G and the knowledge K. In this section, we describe the AMS architecture that handles and maintains all the TAOs.

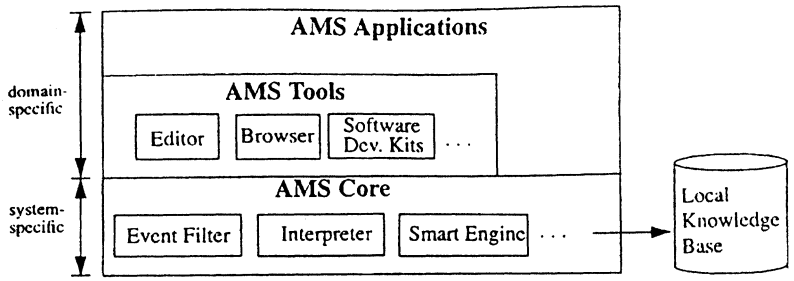

Figure 5. The architecture of AMS.

The AMS consists of two subsystems: a domain-specific part and a system-specific part as shown in Figure 6. The system-specific part performs the generic functions. The application programmers can use the tools and the generic functions to implement his/her own application with TAOs. After the TAOs have been generated by this application and passed to the AMS core, they will be translated, maintained and operated by the system-specific part.

The AMS provides the basic tools upon which users can implement their own applications. A browsing tool is provided for the user to browse the hypergraph G of a TAO. An editor tool is also provided for the user to edit the hypergraph G of a TAO. Other tools allow users to change the system knowledge and build up their own environment knowledge. The template knowledge and private knowledge of TAOs are generated by application-specific tools. For example, we can implement a Smart Multimedia Mail application, which is like a mail system but deals with multimedia mail and also can be associated with private knowledge. In other words, the Smart Multimedia Mail System generates and handles the TAO as a mail object, and all the TAOs generated by this mail system are maintained by AMS, similar to the way files are maintained by an operating system's file manager. The only difference is that AMS will maintain the TAOs as active objects. Another example is a simple multimedia calendar. In this case, the hypergraph part G is not so important, but we can set alarm and a list of tasks to be performed at appropriate times. For example, instead of only beeping, a TAO generated by this calendar can also automatically invoke specified operations.

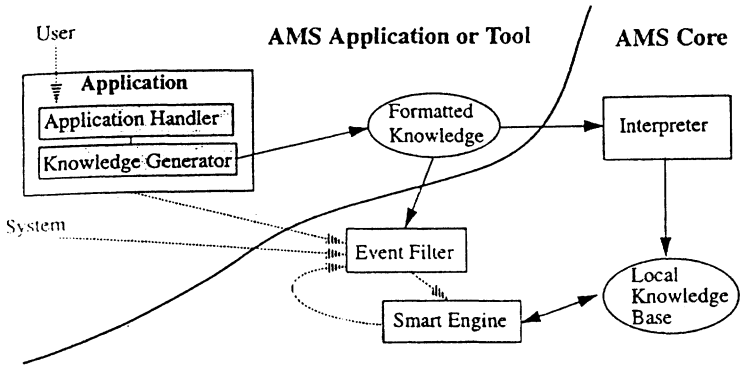

Figure 6. Relation between the AMS application and the AMS Core.

Each AMS tool or user implemented application has the basic components and the relation with the AMS core, as shown in Figure 6. In the above diagram, the line separates the architecture into two subsystems according to the dependence on domain knowledge. The left-hand side is an application-dependent subsystem, including tools provided by AMS and applications developed by programmers. Rectangles represent processes while ellipses represent formatted data. Solid arrows represent knowledge flow while dashed arrows represent event flow. The application-specific Application Handler allows the user to create special purpose TAOs and the Knowledge Generator transforms the knowledge part of these TAOs to the Formatted Knowledge defined in AMS. Therefore, in the right-hand side, the application-independent subsystem AMS itself, when a TAO becomes alive in AMS the Interpreter obtains the formatted knowledge and converts or merges it into the Local Knowledge Base. The local knowledge base is modelled by the Active Index that is a collection of index cells (ICs) [Chang95a] in AMS. At each local AMS system, the Event Filter will monitor the environment, user behavior, specified events, and internal messages, then generate the corresponding messages to the Smart Engine, while the Smart Engine distributes the messages to corresponding ICs in the local knowledge base to invoke the corresponding actions.

4.1 The Local Knowledge Base

The basic concept of AMS is to respond to the environmental changes and take corresponding actions according to user defined knowledge, and to

provide a way such that each TAO has it own private knowledge. In other words, "active" and "private" are the two key ingredients of AMS.

The local knowledge base is defined to be a set of ICs connected by messages [Chang95a]. An IC accepts input messages and performs some computation. It then activates another group of ICs, and posts the output message to these output ICs. If some of these output ICs have already been activated, they may simply accept the output from the current IC. The first output cell that accepts the output message will remove it from the output list of the current cell. After its computation, the IC may remain active (live), or de-activate itself (dead). An IC will also become dead, if it remains inactive for a certain period of time, i.e., if no other cells (including itself) send messages to it. An IC consists of a finite number of ICs. When the IC is in actual computation, it consists of a time-varying collection of ICs in different states, accepting certain input messages and posting output messages to the output lists.

Each IC has two functions: f is an acceptance function that determines when the IC is enabled and ready to fire. Once all the required messages become available, f will remove the messages from the output lists of the message-sending ICs and enable the IC. g is a knowledge function that performs the firing procedure for the IC. Once the f enables the IC, g will take over the control and fire the IC. According to the messages accepted by f, the firing procedure will decide (1) the next state of the IC; (2) how to generate new messages for specific ICs; and most importantly (3) how to perform a specific action sequence.

The IC is the local knowledge base in AMS. And for each TAO in AMS, there are corresponding ICs in the AMS

4.2 Formatted Knowledge for TAOs

According to the definition of an IC, g is the knowledge function in each IC which contains its own finite-state-machine. Basically the g function accepts the input set of messages, computes the next state, generates any new message to other ICs, and performs the corresponding action-sequence. The following BNF definitions show the essential syntax of formatted knowledge used in AMS. By using the hierarchy class name in the < Class Name > of < IC Def > production, we can decide which class of TAO the knowledge function g should be applied to.

```
< Formatted Knowledge >    ::= < Message Def >  < Action Def >  < IC Set Def >
< Message Def >        ::= {EVENT  < Event Name > = < Event Expr. > }*
< Action Def >      ::= {ACTION  < Action Name >  { < Parameter List >  } }*
< Parameter List >          ::= (  < Parameter >  {,  < Parameter >  }* )
```

```
< IC Set Def >      ::= < IC Def > { ; < IC Def > }*
< IC Def >          ::= IC < Type > < ID > < Class Name > < Max Life Time >
                        < FSM >
< FSM >             ::= FSM: < Number of State > < State Trans List >
< State Trans List > ::= < State trans > { , < State trans > }* ;
< State trans >     ::= ( < Current_State > , < Next State > , < Input > , < Output > ,
                        < Action > ) | NIL
< Input >           ::= ( < Message List > )
< Output >          ::= ( { < IC-Message > }* )
< IC-Message >      ::= ( < IC Set > ; < Message List > )
< IC Set >          ::= < IC > { , < IC > }*
< Message List >    ::= < Message > { , < Message > }*
< Action >          ::= ( < Action-process List > )
```

4.3 The Application Handler and Knowledge Generator

For each application, we need an application-specific Application Handler for the end user to work on, and also to provide an easy way to collect the special knowledge given by the end user and this application. After the application creates the TAO, the Knowledge Generator transforms the knowledge gathered from the user to a format suitable for the Interpreter in AMS.

Generally speaking, we can use the multimedia editor tool provided by the AMS to generate the hypergraph structure of a TAO. The knowledge part of a TAO is more application-specific. For example, in a multimedia mail system, the knowledge is about reading, replying, editing, forwarding a mail, etc. While in a medical image system, the knowledge is about diagnosis studies, different image modalities for diagnosis studies, image processing, etc.

4.4 The Interpreter

For each newly created TAO, the Interpreter will transfer the knowledge part K from formatted form to ICs and merge into the local knowledge base. Since the rule set is in a well-defined format, the interpreter can easily generate customized IC dynamically in AMS. In other words, we can use the AMS interpreter to generate a new knowledge function g in an IC dynamically, if necessary.

4.5 The Event Filter

There are numerous events occurring in the system, but for different applications only some events are meaningful while others are not. Therefore, we can use an event filter to filter out events and that can be ignored and allow only meaningful events to enter the AMS.

4.6 The Smart Engine

The Smart Engine maintains the current ICs according to the messages. Since the message is generated by some events, the AMS system is event-driven and can respond to the environmental changes automatically. The message is generated either internally, a result from ICs, or externally, a result from the event filter. The Smart Engine operates as follows: it takes messages and distributes to specific ICs, then checks the corresponding acceptance functions f. If the acceptance function is satisfied, it removes the messages and invokes knowledge function g to fire the IC and performs the action sequence.

When the TAO is not available, it indicates that the corresponding IC is in a dead state, so the Smart Engine needs to remove the IC from the Active Index, and that IC is then withdrawn from the local knowledge base.

5. APPLICATION TO MULTIMEDIA MAIL

Based upon the AMS, we can implement different applications, e.g. smart medical image system, home shopping system, real-estate survey system, etc. In this section, we describe a Smart Multimedia Mail system (SMM) which is based on both AMS and e-mail system.

In SMM, each multimedia mail is a Teleaction Object. SMM provides a simple user interface so that the end user can create his/her private knowledge for individual mail messages. For example, the user can set alarms for his/her important mail (illustrated by the example described in Section 3); collect statistical information; re-route the mail to other recipients; or determine the schedule for a group. The user can also define template mail with associated template knowledge, such as a weekly report, sign-up sheet, etc.

The SMM is implemented in four functional blocks as shown in Figure 7. The Mail-Editing and Mail-Browsing blocks can be implemented using the AMS Editor and AMS Browser tools respectively. Just two blocks are left for the application designer: Mail-Handling and Mail-Knowledge-Editing.

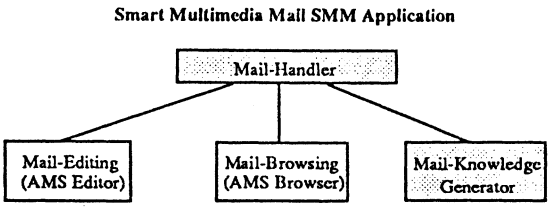

Figure 7. Function diagram of the Smart Multimedia Mail system SMM.

5.1 User Interface of SMM

The SMM begins with an ordinary mail-board. The user ˝ can view and create a multimedia message. When receiving a SMM mail, SMM will pass the private knowledge associated with this mail to AMS and merge it into the reader's local knowledge base. Therefore, whenever an event corresponding to the mail occurs, the IC in AMS will trigger and perform the related actions. When the user views a mail, SMM uses the browsing tool provided by AMS for display and browsing of the mail (a TAO) by the hypergraph layout window and zoom-in windows for detailed browsing, as shown in Figure 8. When the user wants to create a new multimedia mail, SMM also uses an editing tool provided by AMS which supports several ways to get the media data: from file, from database or from live sources.

Besides creating the multimedia mail, the user can also add his/her private knowledge to the mail. The knowledge is formally defined in Section 4, but the user should not be burdened with this detailed format. Instead, the user is provided with an application-specific, easy and friendly user interface. Therefore, SMM needs to provide its own application-specific user interface for gathering the information, using its generator to create formatted knowledge for the private knowledge. Later, the AMS interpreter transfers and merges the knowledge into the local knowledge base. Therefore, in SMM two windows are provided for knowledge acquisition. After the user has composed the new mail content, he/she can add his/her own knowledge to only this mail. Thus, he/she needs to open the knowledge window first, as shown in Figure 9.

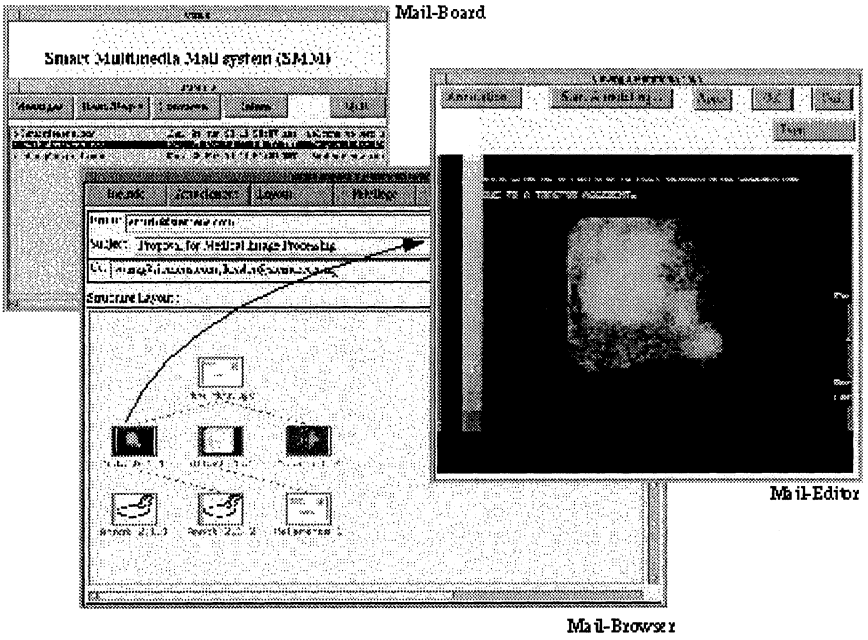

Figure 8. Browsing/Editing in Smart Multimedia Mail system (SMM).

There are two parts for specifying the private knowledge: event sub-panel and action sub-panel. The event sub-panel specifies the events, and the action sub-panel specifies the actions. Each panel can accept more than one statement in `and' connection. In Figure 9, another dialog window is shown to gather such information. Basically, we constrain the knowledge solicited from the user into the following four elements: Who, Doing, To Which Object, and When. (Currently, 'Who' is restricted to have a single value in the conjunctive logic clauses for all event statements in the same event sub-panel.)

The user can use the menu to specify the desired behavior. Moreover, the user can use the mouse (or cursor) to point at objects to specify what specific part of the message he/she refers to. This menu-based interface for specifying the private knowledge of a smart object in SMM is easy to use. All the user needs to do is to select menu items and to point at the objects without worrying about the detailed format of the knowledge. Then, SMM will automatically transform the user's specification into the formatted knowledge by applying the following algorithm.

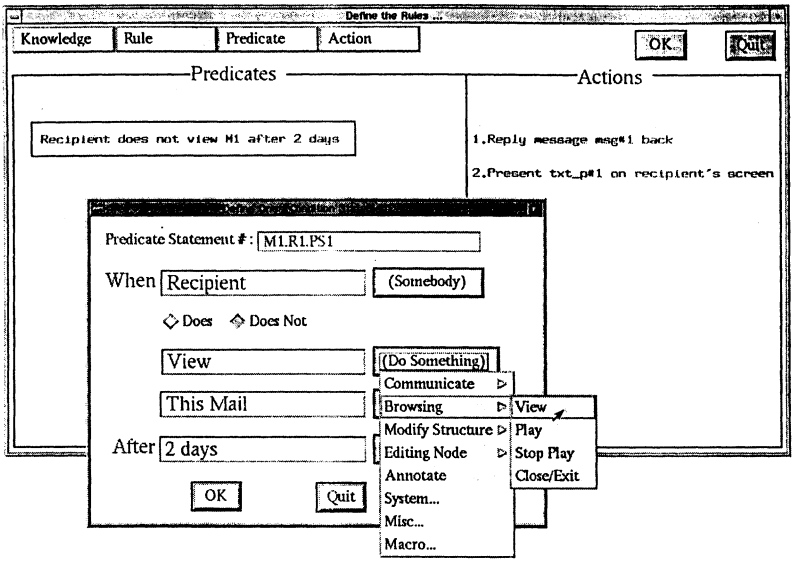

Figure 9. Dialog windows for the private knowledge in SMM.

1. Generate EVENT definitions of SMM.
2. Generate ACTION definitions of SMM.
3. Generate the IC Title with the class name "root.TAO.mail" concatenated with the TAO Id.

/* Assume each sub-panel has one or more statements that are conjunctive */

4. For each statement with "When", use the value of "When" to generate an alarm and place it into the Sender_ALARM set or Recipient_ALARM set according to "who".
5. If Sender_ALARM set is not empty, generate the FSM trans:
   (State0, State1, (SEND), (NIL), (a list of Set_Alarm for all alarms in the Sender_ALARM set))
6. If Recipient_ALARM set is not empty, generate the FSM trans:
   (State0, State1, (ARRIVE), (NIL), (a list of Set_Alarm for all alarm in the Recipient_ALARM set))
7. If both Sender_ALARM set and Recipient_ALARM set are empty, generate the FSM trans:
   (State0, State1, (NIL), (NIL), (NIL))
8. For each event sub-panel /* private knowledge*/
   8.1. For each statement with "When", generate the FSM trans:
   /* setup the alarm & actions*/
   (State1, State1, (event in this statement), (NIL), (Cancel_Alarm of related alarm))
   8.2. For all statements without "When", generate the FSM trans:

/* setup actions */
(State1, State1, (collection of events in all of these statements and all alarms), (NIL),
(all actions in the action sub-panel))

9. generate the final FSM trans:
(State0, Dead, (DEL), (NIL), (NIL))
(State1, Dead, (DEL), (NIL), (Cancel_Alarm of all alarms in all sub-panels))

```
1.  Generate EVENT definitions of SMM.
2.  Generate ACTION definitions of SMM.
3.  Generate the IC Title with the class name "root.TAO.mail" concatenated with the TAO Id.
    /* Assume each sub-panel has one or more statements which are conjunctive */
4.  For each statement with "When", use the value of "When" to generate an alarm and place it into the
    Sender_ALARM set or Recipient_ALARM set according to "who".
5.  If Sender_ALARM set is not empty, generate the FSM trans:
        (State0, State1, (SEND), (NIL), (a list of Set_Alarm for all alarms in the Sender_ALARM set) )
6.  If Recipient_ALARM set is not empty, generate the FSM trans:
        (State0, State1, (ARRIVE), (NIL), (a list of Set_Alarm for all alarm in the Recipient_ALARM set) )
7.  If both Sender_ALARM set and Recipient_ALARM set are empty, generate the FSM trans:
        (State0, State1, (NIL), (NIL), (NIL) )
8.  For each event sub-panel /* private knowledge*/
    8.1.  For each statement with "When", generate the FSM trans: /* setup the alarm & actions*/
              (State1, State1, (event in this statement), (NIL), (Cancel_Alarm of related alarm) )
    8.2.  For all statements without "When", generate the FSM trans: /* setup actions */
              (State1, State1, (collection of events in all of these statements and all alarms), (NIL),
                       (all actions in the action sub-panel) )
9.  generate the final FSM trans:
        (State0, Dead, (DEL), (NIL), (NIL) )
        (State1, Dead, (DEL), (NIL), (Cancel_Alarm of all alarms in all sub-panels) )
```

Figure 10. Heuristic algorithm to generate private knowledge in SMM.

5.2 The Knowledge Generator of SMM

After the user fills out the information in the event sub-panel and action sub-panel, the knowledge generator of SMM will gather all the information and transform it to the formatted knowledge defined in AMS. Following our example in Figure 9, the piece of knowledge is: "If the recipient does not view this mail within 2 days after the mail arrives, then send a message back to the sender and beep a message to the recipient". By applying the heuristic algorithm in Figure 10, the corresponding pieces of formatted knowledge are generated as shown below:

```
EVENT ARRIVE =  ...
EVENT VIEW =  ...
EVENT ALARM2 = ...
EVENT DEL =    ...
. . .
```

```
ACTION    Set_Alarm(own_IC_Id, duration, event_label, ...)
ACTION    Cancel_Alarm(IC_Id, event_label, ...)
ACTION    Reply_Message(user_Id, message_no, ...)
ACTION    Beep_Message(user_Id, message_no, ...)
. . .
IC    M1 #1234 "root.TAO.mail.#1234" INFINITE_TIME
  FSM : {State0, State1, Dead}
     /* setup alarm when arrival in recipient */
     (State0, State1, (ARRIVE), (NIL),
     (Set_Alarm(own_IC_Id(), 2_days, ALARM2, ...))
     /*Cancel alarm when recipient views it */
     (State1, State1, (VIEW), (NIL), (Cancel_Alarm(own_IC_Id(), ALARM2, ...)))
     /* Reply and beep if expired */
     (State1, State1, (ALARM2), (NIL), (Reply_Message(sender_Id(), mesg1, ...),
     Beep_Message(own_Id(), mesg1, ...) ))
     (State0, Dead, (DEL), (NIL), (NIL))
     (State1, Dead, (DEL), (NIL), (Cancel_Alarm(own_IC_Id(), ALARM2, ...) ))
```

6. APPLICATION TO MULTIMEDIA INFORMATION RETRIEVAL

To retrieve information in the hyperspace which is represented by a hyperstructure (i.e. a hypergraph structure), we can associate an IC with every recently accessed node in this hyperstructure. Thus the IC is a finite set of recently accessed nodes. The IC can be constructed as follows: It accepts a query and activates the adjacent ICs, and in turn posts queries to them. The action performed is to pre-fetch information items satisfying the query. A further refinement is to pre-fetch information items above a certain size. The justification is that we need only pre-fetch large information items, small information items need not be pre-fetched. At the University of Pittsburgh, we have added the IC to Mosaic, creating a new version called Mosaic-IC which has pre-fetching capabilities. An example of Mosaic-IC is illustrated in Figure 6 of Chapter 5, where the background window on the right displays the trace of executions of the IC, and the action_icons in the upper-right corner show the actions performed.

From the above application examples, it can be seen that the two important features we introduced in the Active Multimedia System - `active' and 'private' - are both realized by the Active Index. In AMS, the users can specify their private knowledge and then combine that with the system's knowledge, resulting in greater flexibility in AMS's adaptive behavior.

The `private' knowledge means polymorphism - certain objects can obtain reactions different from reactions to other objects even in the same environment. For example, in the AMS mail system described in Section 5, the users can specify their private knowledge such as the importance of different message classes. Therefore, the sender specifies his/her private knowledge (the importance of message classes) so that if the recipient forgets to view a particular message, a reminder will show up on both the sender's and the recipient's screen. Another interesting example is the inclusion of different levels of pre-fetching methods in Mosaic-IC, which is the Mosaic browser equipped with ICs. The AMS will provide two basic levels of pre-fetching methods. For a particular application, users can add their own pre-fetching methods based on their special considerations.

Basically, the Active Index maintains the `active' knowledge of AMS. In order to combine private knowledge with system's knowledge, in the Active Index we can divide the ICs into groups to form a hierarchy. In this hierarchy, one class of ICs can share the same methods. Messages sent to higher level ICs will be handled by the higher level methods. Only when there is no higher level method, will the message be sent to ICs at lower levels and handled by lower level methods.

Going back to the previous example of customized pre-fetching methods in Mosaic-IC, as shown in Figure 11, there are different classes of ICs in the hierarchy. XIC1 uses the simplest pre-fetching method of sequentially retrieving objects. XIC2 uses a smarter pre-fetching method by consulting both the current system profile and the user profile. XICT is designed to perform catalog searching, and can pre-fetch information from a special catalog. Therefore, most users using the Mosaic-IC will only need one or two levels of pre-fetching. But once the users get into the catalog searching application, they may need one additional level of pre-fetching.

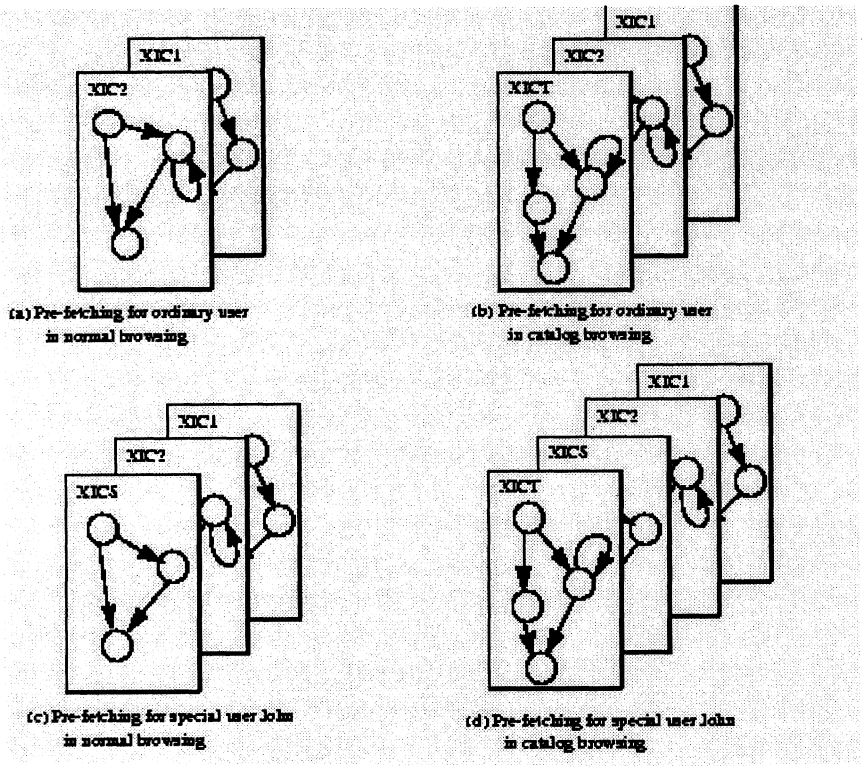

Figure 11. Different prefetching situations.

For different applications, different ICs from the hierarchy can be used. XIC1 pre-fetches in sequence, XIC2 pre-fetches by user/system profiles, XICS pre-fetches based upon the result of learning from users' past behaviors, while XICT pre-fetches by special knowledge in catalog searching.

Of course it is possible to have more levels of pre-fetching methods. For example, the AMS can use a learning monitor to observe the Mosaic-IC users' behavior. Then the AMS can develop a specialized IC at a new intermediate level with a customized pre-fetching method. For instance, XICS is for a particular user such as John who checks the technical reports on multimedia database very often using Mosaic-IC. Then John's normal navigation in Mosaic-IC will use three levels of pre-fetching methods: XIC1, XIC2, and XICS, where XICS has the highest preference in `multimedia'. But when John steps into the catalog searching application, then the pre-fetching methods may include four levels: XIC1, XIC2, XICS, and XICT. Combined, the pre-fetching will prefer the current interests in catalog

searching, then in `multimedia' information, and then according to user/system profiles and finally the sequential accessing of all objects.

In implementation, the message M is sent to the hierarchical group of ICs, not to an individual in the group. With a tag in the message M, the message M is either a normal message or a broadcast message. For the normal message M sent to the hierarchical group, the highest level will catch this message first. If the highest level cannot handle this message, it is passed to the next higher level until one IC catches it, and finally to the lowest level IC. For the broadcast message, it will be passed to the next level IC whether the current IC catches it or not. But in both cases, the message passing is in a sequential order, not in a non-deterministic order, as shown in Figure 12. For example, a KILL message for the hierarchical group of pre-fetching should be a broadcast message, while the PROFILE_CHANGED message could be a normal message that is caught only by XIC2.

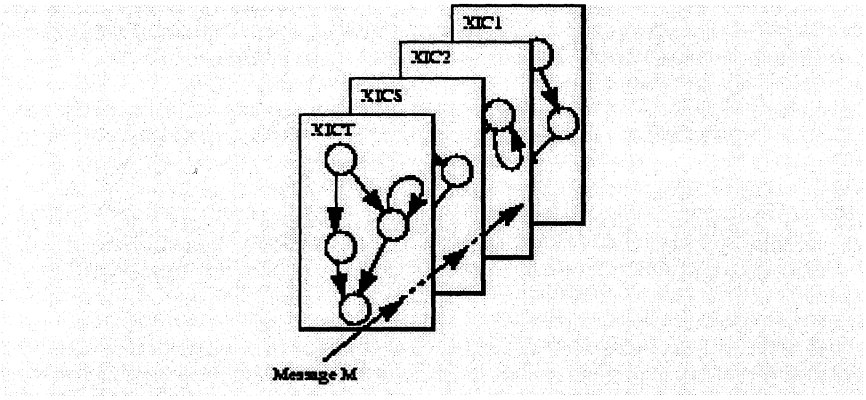

Figure 122. Message passing through a hierarchy of ICs.

7. DISCUSSION

A desired requirement for multimedia applications is that the user be able to interact with multimedia objects that are aware of environmental changes and moreover are able to dynamically incorporate unanticipated information to better react to environmental changes.

In this chapter we have presented the Teleaction Object model for the design of an Active Multimedia System. This model emphasizes a unified approach for the modelling of multimedia applications, presentation and communication, as a collection of interacting objects. Teleaction Objects are formally specified as (G, K) pairs. With the types of node, media, and link in

the hypergraph G, we can design algorithms for the efficient communication and effective presentation of multimedia information. Using multiple levels of knowledge realized by ICs, the user could customize the AMS and also apply private knowledge to certain group of TAOs.

Our approach has the following advantages as compared to the traditional AI approach: (1) The AMS system is not a rule-based system. It is based on the IC technique that will dynamically modify itself to perform various operations. (2) The TAO incorporates hypergraph structures that represent domain knowledge not easily captured by expert system rules. (3) The IC technique enables the system to localize a small set of rules in its operations. The AMS system, therefore, is more efficient than a general purpose expert system. ICs also share some characteristics of intelligent agents [ACM94], but the fundamental difference is that an IC is a data structure to facilitate information access and knowledge processing. We can use an IC to realize B-trees and other conventional data structures efficiently. An Active Index may have millions of ICs, whereas agent-based systems typically have at most hundreds of agents.

A prototype Active Multimedia System (AMS) for Smart Multimedia Mail application (SMM) has been implemented on SUN workstations. Currently, our implementation effort for AMS is taking a direct approach. It has been our concern for the standardization regarding the implementation of AMS. As mentioned in Chapter 2, an ISO standard for programming environments for the presentation of multimedia objects, called PREMO [Herman94], has drawn a lot of attention. PREMO addresses the issues of configuration, extension, and inter-operation of and between PREMO implementations. From its conceptual framework, PREMO is based on an object model in which object operations can be synchronous, asynchronous, or sampled. Besides active objects, events are used as the basic building block for its event model. Standards like PREMO will eventually provide a standardized development environment for active multimedia systems such as AMS.

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.

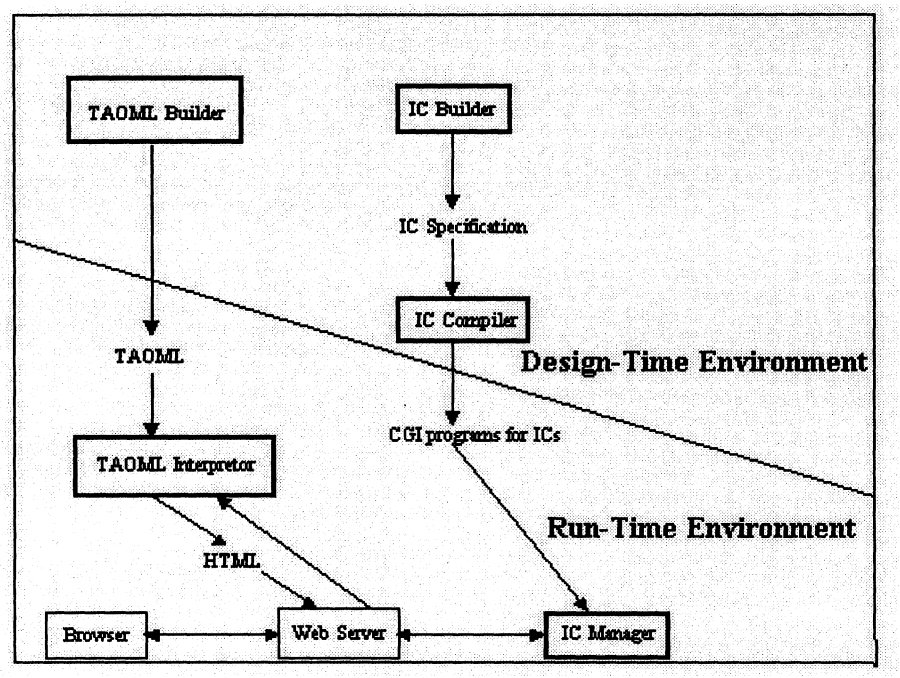

Figure 1. MICE Tools.

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.

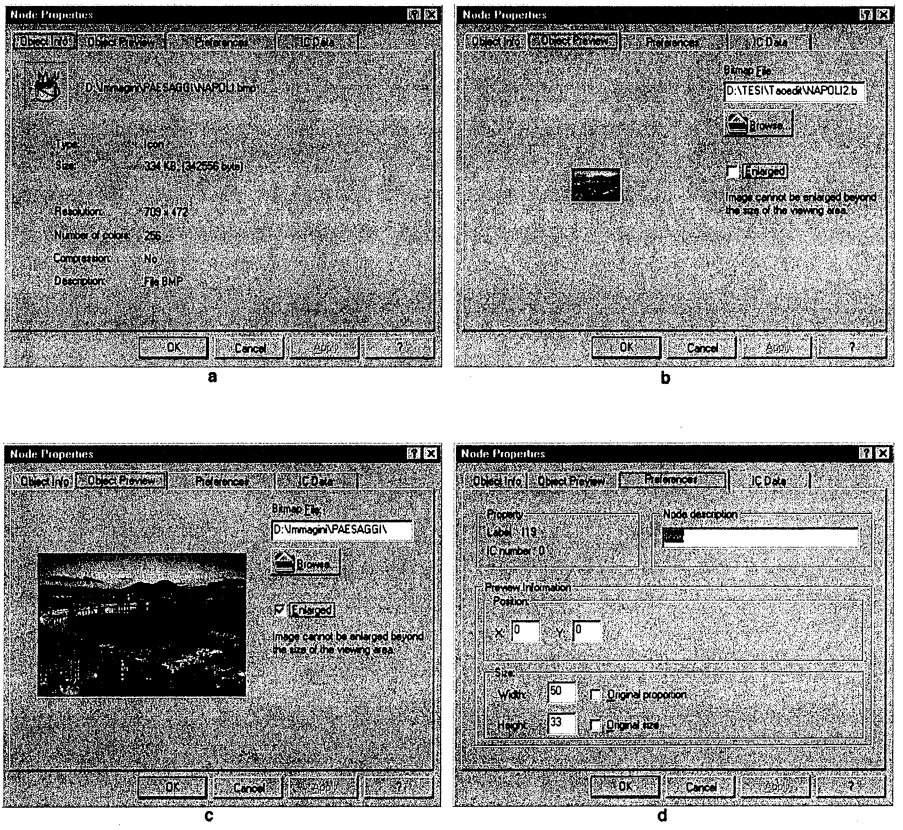

Figure 2. Tabbed Dialog Showing Node Properties.

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.

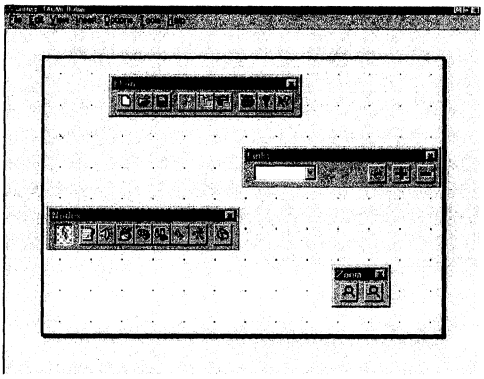

Figure 3. TAOML Builder Toolbars.

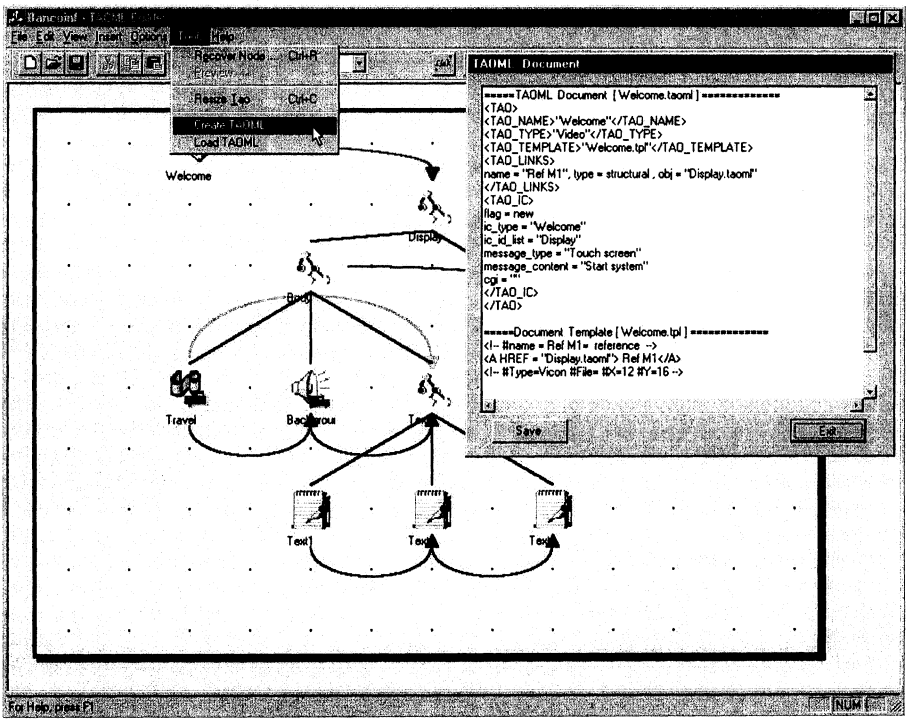

Figure 4. Figure 4 - Hypergraph and Matching TAOML.

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.

Chapter 8

Pragmatics: Prototyping Multimedia Applications

MICE is a multimedia development environment for the rapid prototyping of multimedia applications. The traditional "waterfall" software life cycle model is depicted in Figure 1.

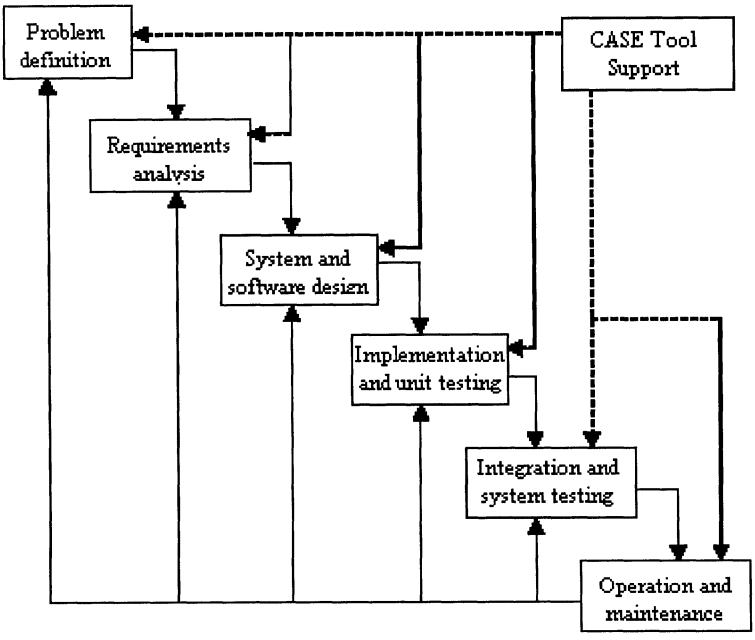

Figure 1. Traditional "waterfall" software life cycle model

This traditional software life cycle is appropriate for traditional application software development. Multimedia applications, on the other

hand, place strong emphasis on evolutionary content development. The rapid prototyping model is depicted in Figure 2.

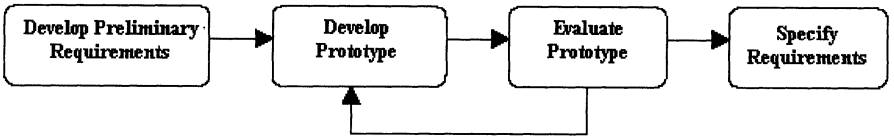

Figure 2. Rapid prototyping model for software development.

MICE is an application software development environment supporting rapid prototyping. In this chapter the details of MICE will be explained.

1. HOW TO BUILD A MICE APPLICATION

MICE provides a logical way in building a multimedia application on a workstation or a PC. From the web site www.cs.pitt.edu/~chang, by following the links to multimedia software engineering courseware, you will be led to the following directories that contain the essential components of MICE:

- IC_Builder/ the files you need to run the IC_Builder on PC
- IC_Compiler/ the files to run the IC_Compiler
- IC_Manager/ the files needed to compile the IC_Manager
- IC_Taoml/ the interpretor to translate .taoml pages to .html pages

The seven steps to build a MICE application in a project directory such as IC_Work/ are described below:

Step 1. Download the IC_Builder to your PC, unzip and install it under Windows. Use IC_Builder (see Section 2) to draw each index cell and create the .in file for each ic. Use capital letters for the *.in files such as XIC.in, DIC.in, etc. The IC_Builder will also create the ic.dat file. You can also create *.in and ic.dat manually without using IC_Builder.

Step 2. Provide one action file for each action defined in the ic's. An action file contains the corresponding C function for each action and should be copied to the IC_Work/source/ directory.

Step 3. Copy *.in files to IC_Work/ directory and ic.dat file to IC_Work/source/ directory. Copy all files from IC_Compiler, IC_Manager and IC_Taoml to IC_Work/source/ directory. If necessary, modify the ic.dat file. Use IC_Compiler icc to generate the source files. There are six files generated by IC_Compiler:

actions.c ic_func2.c ic_func3.c app.h fuzzy.h db_def.h

Step 4. Use command "make -f makefile.maincgi" to make main.cgi that is the cgi program that your application will need. Therefore, you may call it main.cgi. The IC_Manager becomes part of main.cgi so any message to an ic will be sent to this main.cgi.

Step 5. Use command "make -f makefile.intercgi" to make inter.cgi that is the cgi program to access a taoml page. main.cgi and inter.cgi should be copied to application directory IC_Work/ so that the home page can use inter.cgi to access a taoml page such as tao_1.taoml. The link has the following form: <a href="inter.cgi?tao_name=tao_1.taoml">.

Step 6. Design the home page index.html for the application and put it in directory IC_Work/. This home page should have a cgi link to a taoml page such as tao_1.taoml. tao_1.taoml and its associated template page tao_1.tpl should be in the sub-directory IC_Work/TAOML/. Indeed, all taoml pages and tpl pages should be in the sub-directory IC_Work/TAOML/. To create the taoml pages, you will use an extended html syntax to specify the TAOs and how they are structured and activate ic using the cgi program, which is main.cgi. If you want to refer to your own cgi programs, they can be mentioned in the tpl pages. To sum up, there are three types of pages:

- html page index.html uses cgi program inter.cgi to link to tao_1.taoml
- taoml page tao_1.taoml uses cgi program main.cgi to activate an ic tao_1
- tpl page tao_1.tpl uses customized cgi program to do special tasks

Step 7. Now you are ready to run your application. Use a browser to enter application's home page IC_Work/index.html.

Notes:

- For detailed step-by-step instructions, see Section 6, MICE Application Development Steps.
- Program compilation must be done on the same type of computer system as the server.
- In IC_Compiler directory, the action template is action.tpl and the customized action functions are a1.c, a2.c, ..., etc. which correspond to the actions a1, a2, ..., etc.

2. IC BUILDER

The IC Builder is a PC-based tool to help the user define active index cells. Once an ic is defined, the IC Builder creates a formal specification file *.in such as ic1.in. After all the ic's 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 ic specification files *.in, on the other hand, become the input to the customized IC Manager.

The Symbolic IC_Builder Version 2.0 has the following features:

- There is no need to specify the message ID and action ID any more, i.e. messages and actions are directly represented by symbolic names.
- The ic.dat file will be automatically generated.
- Simple project management for all the ICs in the project.

The following steps will create the ICs and .in and ic.dat files:

Step 1. Create a project directory, which will contain all the project files later, such as c:\icb\hw4\

Step 2. In the IC Builder menu bar, find the 'simulation' menu. Select the 'options' menu item. This will bring up a dialog asking you to specify the executable application file (the customized IC Manager) and the message input file. For example, if your customized IC Manager is an executable file called wag.exe, then you first enter:

c:\icb\hw4\wag.exe

and then you enter:

c:\icb\hw4\mag.in

which means the message input file is msg.in. These two inputs are served as the simulation purpose in IC_Builder. But they are also used to determine the project directory, which will be subtracted from the executable file path. This means all the files generated by IC_Builder will be put in this directory.

Step 3. Also in the 'simulation' menu, select the 'project' item to define the project files in the project. You could add or remove the files from the project. Please use the name of IC file, don't use the .in name. For example, if your project contains three ics: WAG, BBC, LBC, then the project files could be:

WAG.gra BBC.gra, LBC.gra

Step 4. Use the IC_Builder to create ICs. The input message and output message specification dialogs are explained later in this section.

Step 5. After you have created all the ICs, click the 'export' button to export the .in files and ic.dat. IC_Builder will create .in file for each IC, the name of .in file is same as its graphic file.

The key features of the IC_Builder are described below:

2.1. Define a Project

2.1.1 Create a Project Directory as Your Work Space: Create a directory in the disk, which will contain all the project files later. For example, c:\icb\hw4\.

2.1.2 Specify Project Files: In the IC Builder menu bar, find the 'simulation' menu. Select the 'options' menu item. This will bring up a dialog which ask you to specify the executable file for the customized IC Manager of the application and the external input message file if necessary. Please be

sure to specify the complete path of the project directory so that any file generated by IC_Builder will be put in this directory.

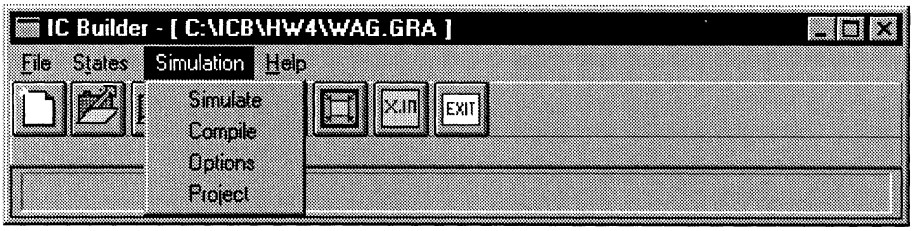

Figure 3. How to specify project files.

For example, if your customized IC Manager is an executable file called wag.exe, then at the first line input the following:

c:\icb\hw4\wag.exe

and the second line can be something like:

c:\icb\hw4\mag.in

which means the external input message file is msg.in.

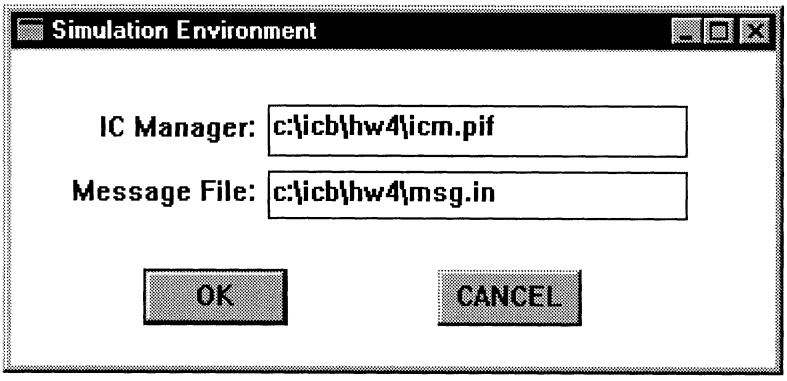

Figure 4. How to specify IC Manager and message file.

2.1.3 Specify IC Types used in the Project: Also in the 'simulation' menu, select the 'project' item to define the project files in the project. You could add or remove the files from the project. Please use the name of IC file, don't use the .in name. For example, if your project contains three ics: WAG, BBC, LBC then the project files could be: WAG.gra BBC.gra, LBC.gra.

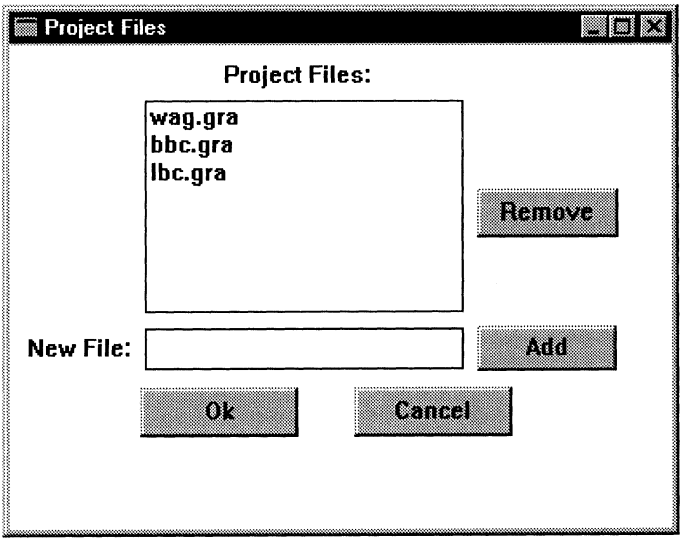

Figure 5. The project files.

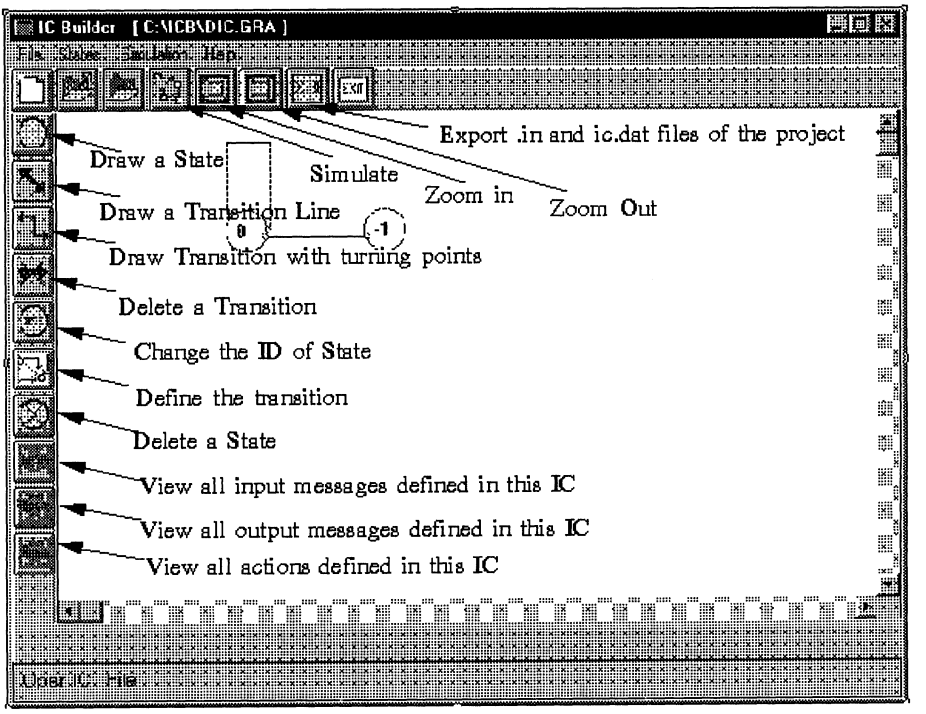

Figure 6. The IC Builder's tool bar.

2.2. Define IC Types

2.2.1 Draw a State: Click (press left mouse button) the icon in the tool bar, then point the cursor at the desired position, press left button. The state will be numbered automatically.

2.2.2 Delete a State: Click (press left mouse button) the Delete_State icon in the tool bar, then move the cursor within the state which you want to delete, press left button.

2.2.3 Move a State: Currently, there is no way to move a state to a different position, you have to delete and draw a new one to achieve the reposition of the state.

2.2.4 Change the ID number of the State: Click (press left mouse button) the Change_ID icon in the tool bar, then click on the state you want to change. A dialog will appear to let you enter the new number of the state.

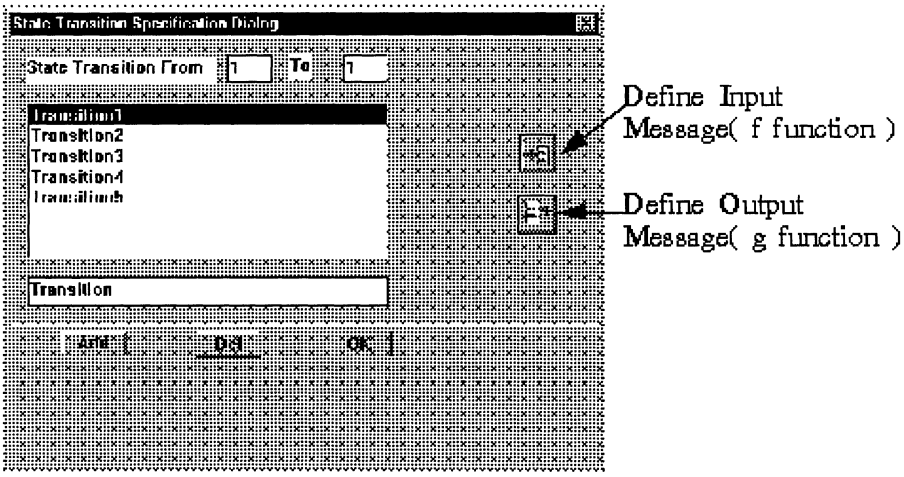

Figure 7. How to define a transition.

2.2.5 Draw a Transition: There are two ways to draw a transition between states. One way is to draw a straight line, the other is to draw a line with turning points. Either way you should first click the icon, then move the cursor to the transition start position on one state, then click. Then you can move the cursor to the next position, click, and so on (if you are not using the draw straight line icon). Finally you double click the left button to select the end position of the transition. Be aware that the start and end positions should always be on the edge of the state. The start position is marked as a small green rectangle, and the end position is marked as a red rectangle.

2.2.6 Delete a Transition: Select the Delete_Transition icon in the tool bar, move the cursor to the start position of the transition, then click the left button.

2.2.7 Define a Transition: Click the Define_Transition icon in the tool bar, then move the cursor to the start position of one transition, click the left button. A dialog like above will appear on the screen. This dialog lets you to add as many transitions between two same states as you want. Click the two buttons on the right of the dialog to further define the input or output message of one transition between the two states.

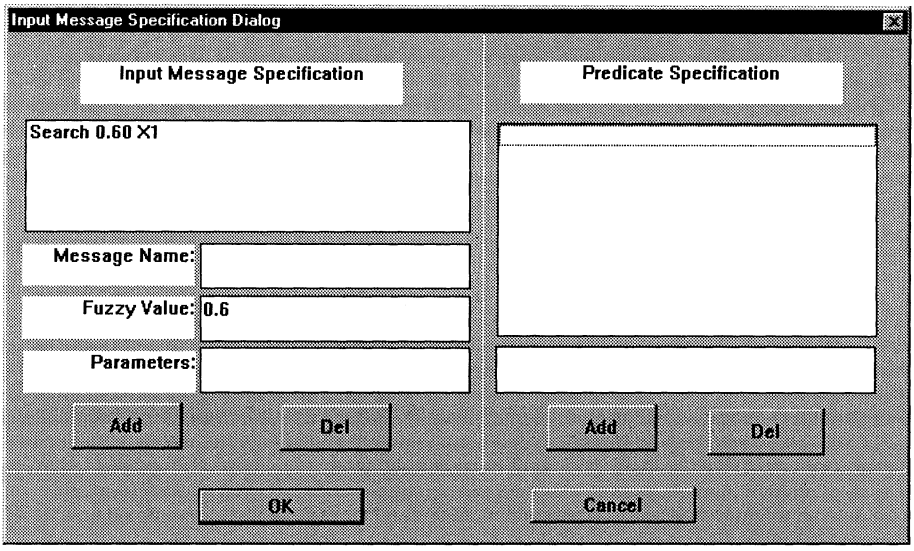

Figure 8. How to define the input message.

2.2.7.1 Input Message Specification Dialog: As you click the Define Input Message button, the Input Message Specification dialog will prompt, as the figure below. There are two columns in the dialog, the left one is used to specify input message's name, parameters. The right column is used to define the predicate for input messages. You can add or delete the input message and predicate. The format of the input field "Parameters" will be explained later in Section 2.3. Notice the message name is case insensitive.

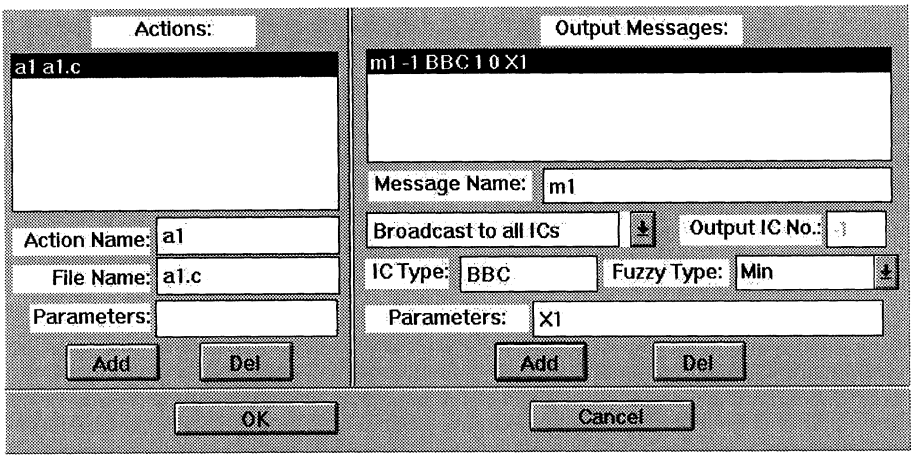

Figure 9. Output message specification.

2.2.7.2 Output Message Specification: 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. The format of the input field "Parameters" in the action and output message will be explained later in Section 2.3. 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 which will be activated when this message comes. Please notice you must specify the IC type in the field "IC type".
- 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 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".

2.2.8 Export the IC diagram: The IC diagrams definition need to be transformed to so called .in file when it is used as the input of the IC_Manager. Click the Export icon in the tool bar to export 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 IC compiler.

2.3. The format of the parameter in the Message Definition Dialog is given below in BNF syntax:

<para_list> ::= <para_list>'|'<item>
<para_list> ::= <item>
<item> ::= <const> | <var> | <func>
<const> ::= I<integer> | F<float> | S<string>
<var> ::= X<digit> | Y<digit> | Z<digit>
<func> ::= G<digit>'('<func_para_list>')' | H<digit>'('<func_para_list>')'
<func_para_list> ::= <f_para_list> | NULL
<f_para_list> ::= <f_para_list>','<item>
<f_para_list> ::= <item>
<digit> ::= '0'..'9'

For example, X1|Y1|G7(X2, Y2) in the field parameters means that X1 and Y1 are variable parameters and G7 is a function parameter which has two variable parameters X2 and Y2. I25|F1.2|Sfire means that there are three constant parameters: an integer 25, a floating point 1.2 and a string "fire".

2.4. A sample .in file is given below:

```
0       // current state
0       // next state
1       // 1 input message(s)
10:0,Y1|Y0 // message start_prefetch with 2 parameters
0       // no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
3       // 3 action(s)
11       // action "issue_proc"
12       // action "set_pid"
14,Y1|Shelp|H0(Y0) // action "compute_schedule" with 3 parameters
0       // current state
0       // next state
1       // 1 input message(s)
```

```
11:0     // message "end_prefetch"
0        // no. predicate
0        // 0 output ic(s)
0        // 0 output message(s)
1        // 1 action(s)
15       // action "set_pid_null"
0        // current state
-1       // next state
1        // 1 input message(s)
12:0     // message "kill_prefetch"
0        // no. predicate
0        // 0 output ic(s)
0        // 0 output message(s)
1        // 1 action(s)
13       // action "kill_proc"
```

3. IC COMPILER

The IC Compiler accepts an input file that characterizes an application and generates the customized source code of the IC Manager. The default input file is ic.dat produced by the IC Builder. The IC Compiler icc can be recompiled using the make file "makefile.icc".

Usage: icc [-d] input_file

The flag -d generate source codes with embedded debugging messages

Input of icc: The input_file specifies the characteristics of the application. The default input_file is ic.dat.

Output of icc: app.h, fuzzy.h, db_def.h, actions.c, ic_func2.c, and ic_func3.c.

Format of the input_file: Each definition type header must be prefixed by "$". All definition lines follow their definition type header without prefixed by any special character. A definition type must end with "%". A comment line must begin with "//". A space line is allowed.

The IC Compiler supports the following definition types:

(1) Header "$MESSAGE" defines input and output messages of IC with definition format:

message_name/message_id

This definition type is to generate message definitions in "app.h", message array msg[] in "fuzzy.h", and function decode_msg() in "ic_func3.c".

(2) Header "$INCLUDE_FILE" allows the user to adds including files to app.h with definition format:

file_name

(3) Header "$ACTION/AUTO_GEN:YES|NO" defines actions of IC with definition format:

action_name/action_id[/function_name[/file_name]]

This definition type is to generate action definitions in "app.h", function do_actions() and all action functions in "actions.c". If "actions.c" exists, the old "actions.c" will be moved to a backup file "actions.b*", for example, actions.b0, actions.b1....

If AUTO_GEN equals YES, actions.c will be automatically generated by collecting specified file_names; otherwise, all the file_names will be ignored and the user has to provide an actions.c by himself.

If AUTO_GEN equals YES but the file_name is not specified, a default template of the corresponding function will be inserted into the actions.c instead.

(4) Header "$IC_ID" defines IC_IDs with definition format:

name_of_ic_id/number

This definition type is to generate ic_id definitions in "app.h" and function decode_ic() in "ic_func3.c".

Note: ic_id EXTERNAL has been defined as -1 in "ic.h".

(5) Header "$MUST_FUNC/AUTO_GEN:YES|NO" defines functions that are necessary in IC Manager with definition format:

func_group_name[/file_name]

This definition type is to generate all functions of file "ic_func2.c". If "ic_func2.c" exists, the old "ic_func2.c" will be moved to a backup file "ic_func2.b*", for example, ic_func2.b0, ic_func2.b1....

If AUTO_GEN equals YES, ic_func2.c will be automatically generated by collecting all functions in specified file_names; otherwise, all the file_names will be ignored and the user has to provide an ic_func2.c by himself. If AUTO_GEN equals YES but file_name is not specified, a default template of the corresponding functions will be inserted into ic_func2.c in stead.

(6) *.tpl are the templates for $MUST_FUNC. The templates: out_msg.tpl, predicat.tpl, inter_mm.tpl, and func_var.tpl are default templates for functional groups FILL_OUTPUT_MSG_GROUP, PREDICATE_GROUP, INTERNAL_MM_GROUP, and USER_DEFINE_FUNC_VAR_GROUP, respectively. The customized

functions should be called out_msg.c, predicat.c, inter_mm.c, and func_var.c.

A default template for each group contains:

```
FILL_OUTPUT_MSG_GROUP: fill_content(), fill_itype()
PREDICATE_GROUP: pred_match()
INTERNAL_MM_GROUP: dump_internal_mm(), init_mm(),
        save_mm(), restore_mm()
USER_DEFINE_FUNC_VAR_GROUP: userdef_f(), userdef_v()
FILL_OUTPUT_IC_GROUP: find_ic()
```

It is recommended to copy and modify the default template functions for each function group.

(7) action.tpl is the template for user-supplied action functions, one for each action. The customized actions are stored in separate files such as a1.c, ..., a5.c.

(8) Header "$THRESHOLD" defines thresholds for fuzzy computation with definition format:

```
ic_type/fuzzy_number
```

Two arrays ic_type and threshold will be generated in "fuzzy.h".

Note: Definition type "$THRESHOLD" is necessary for fuzzy IC.

(9) Header "DB_ACCESS" defined the access of database with definition format:

```
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 is all attributes of the specified tables. If condition is "NULL", it means that the generated SQL has no condition. The definition will create a SQL command for the specified database:

```
SELECT attributes
FROM tables
WHEN condition
```

File db_def.h will be generated.

(10) Use makefile.icc to compile the IC Compiler. Usage:

```
make -f makefile.icc
```

Note: All definition types must always be specified, except $IC_ID.

4. THE IC MANAGER

This directory contains programs to use fuzzy IC manager.

Make File: The following make files contains paths that need to be changed, if necessary.

Makefile.maincgi: generate main.cgi that will trigger ICs.
Makefile.showcgi: generate show.cgi that will display all ICs.
Makefile.clearcgi: generate clear.cgi that will clear all ICs.

Header Files

I. Core Part: modules in this part should not be modified

ic.h: header file (constants and data structures) of the ic manager

II. Application Dependent Part: constants and data structures in the header files in this part should be customized according to the application.

app.h: constants and data structures of your application
fuzzy.h: the header file for message codes and thresholds of ic types
It is included in fuzzy.c.
mm.h: structures for internal memory of ic's
If you define mm.h, define the including of mm.h in ic.dat so that app.h will include mm.h.

C Files

I. Core Part: modules in this part should not be modified

main.c: the web-based driver functions.
ic_manager.c: ic manager
ic_functions.c: functions called by ic manager
util.c: functions to create C structures for f, g functions and messages to dump the content of various structures.
fuzzy.c: functions to implement fuzzy computation.
clear.c: a function to clear all ICs.
show.c: a function to display all current ICs.

II. Application Dependent Part: modules in this part should be customized according to the application. Examples and/or templates are provided for functions in each module.

driver.c: the text-based driver program
ic_func2.c: application dependent, but necessary functions
ic_func3.c: application dependent decoding functions
ic_state.c: functions to save and restore states of ic's.
actions.c: action functions

Input Files

*.in: f, g functions of index cell
fuzzy.dat: (not to be modified): fuzzy computation table.
ic.dat: IC specification for IC Compiler
*.tpl: template input file to IC Compiler (user can modify it)

Manuals
f.g format: f, g function format
output_ic_msg: the usage of "output ic" and "output msg"
in fg function format
ic_prog: manual for ic programming
msg.format: FAKE external message format for the testing of
your active index system

5. TAOML

This directory contains the TAOML interpreter. Makefile.intercgi will generate an executable "inter.cgi". Makefile.inter will generate an executable command "inter".

6. MICE APPLICATION DEVELOPMENT STEPS

The MICE Application Development Steps are as follows:
Step 1. Download IC_Builder to PC and use it to create *.in, *.gra and ic.dat files.
Step 2. Upload *.in and ic.dat to your working directory. Create two sub-directories called "source" and "TAOML". Leave *.in in this directory, and move ic.dat to "source" directory.
Step 3. Change to source directory and do the following:
Step 3.1. Copy all the files from the three
directories IC_Compiler, IC_Manager and IC_Taoml.
to this "source" directory.
Step 3.2. Move ic.dat and action*.c files into the source directory.
Step 3.3. Invoke IC Compiler by typing:
icc ic.dat
Step 3.4. Use the makefiles to make main.cgi and inter.cgi:
make makefile.maincgi
make makefile.intercgi
Step 3.5. Move main.cgi and inter.cgi programs to parent directory.
Step 4. Create index.html that is the home page of your application.
It should invoke inter.cgi to go to another taoml page.
Step 5. Change to TAOML directory and do the following.
Step 5.1. create *.taoml pages which should invoke main.cgi to activate ic's.
Step 5.2. create *.tpl pages which should invoke inter.cgi to access

Another taoml page, or invoke customized cgi to do special processing.

Step 6. You are now ready to test the application. Use a web browser to access your application's home page index.html.

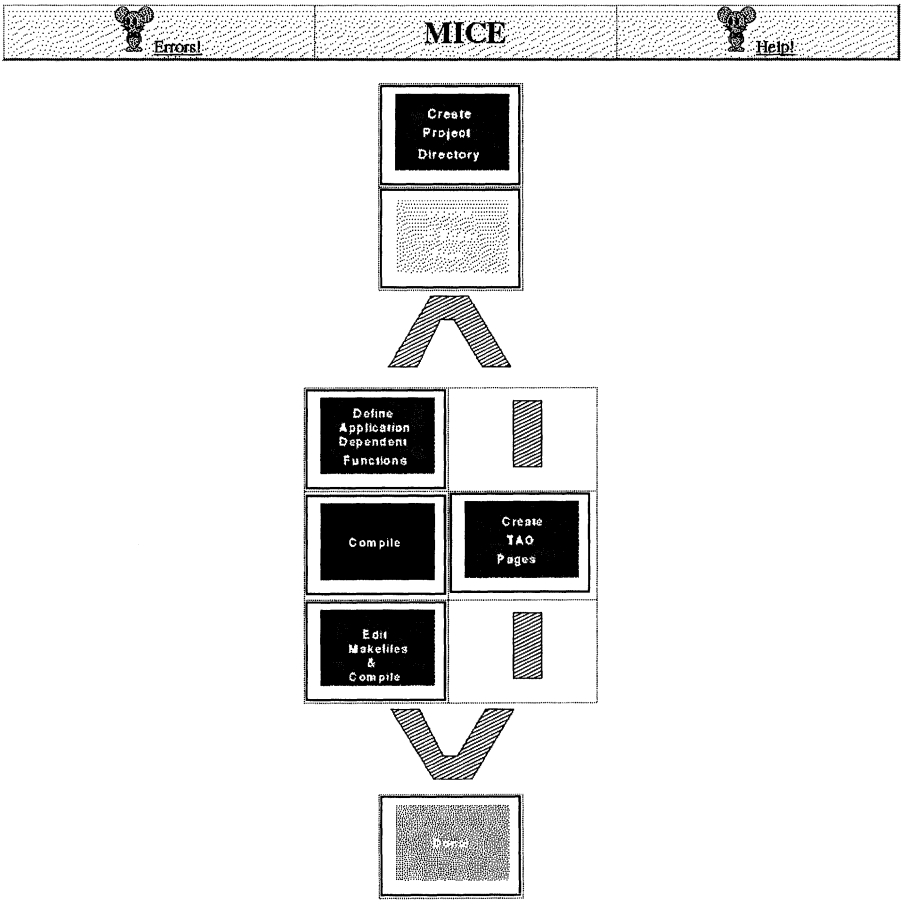

Figure 10. A visual diagram for MICE.

7. VISUAL INTERFACE FOR MICE

The visual interface for MICE is intended for the end user, so that the user does not have to memorize the development steps and the details of multimedia application development using MICE. As illustrated in Figure 10, VISUAL MICE provides a visual diagram. All the user has to do is to

follow the visual diagram and provide the appropriate information at each step.

8. MICE APPLICATIONS

The MICE design environment can be applied to designing all kinds of active multimedia information systems. In what follows, we describe a recent application to active medical information system design [Chang98a, Chang98b].

To accomplish the retrieval, discovery and fusion of medical information from diverse sources, an active medical information system capable of retrieving, processing and filtering medical information, checking for semantic consistency, and structuring the relevant information for distribution is needed. We have developed a framework for the human- and system-directed retrieval, discovery and fusion of medical information, which is based upon the observation that a significant event often manifests itself in different media over time. Therefore if we can index such manifestations and dynamically link them, then we can check for consistency and discover important and relevant medical information.

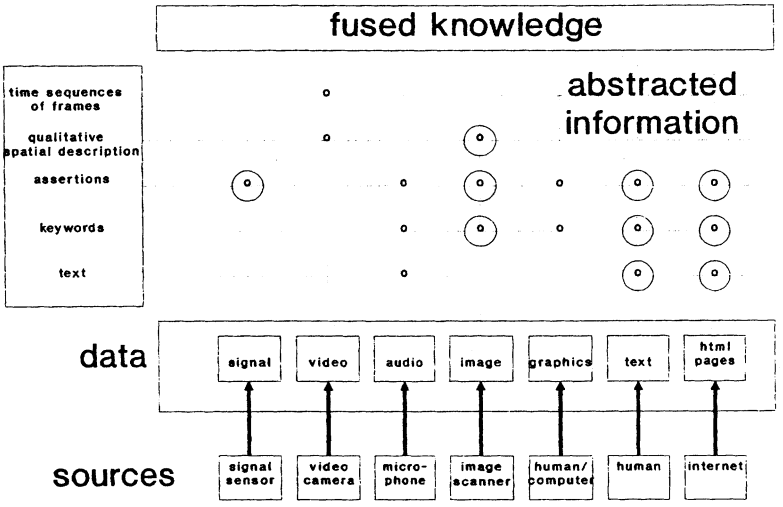

Figure 11. A framework for information and knowledge fusion.

This dynamic indexing technique is based upon the theory of active index. A powerful newly developed artificial neural network is used for the discovery of significant events. An experimental system was implemented, and MICE was used as the prototyping environment to prototype AMIS.

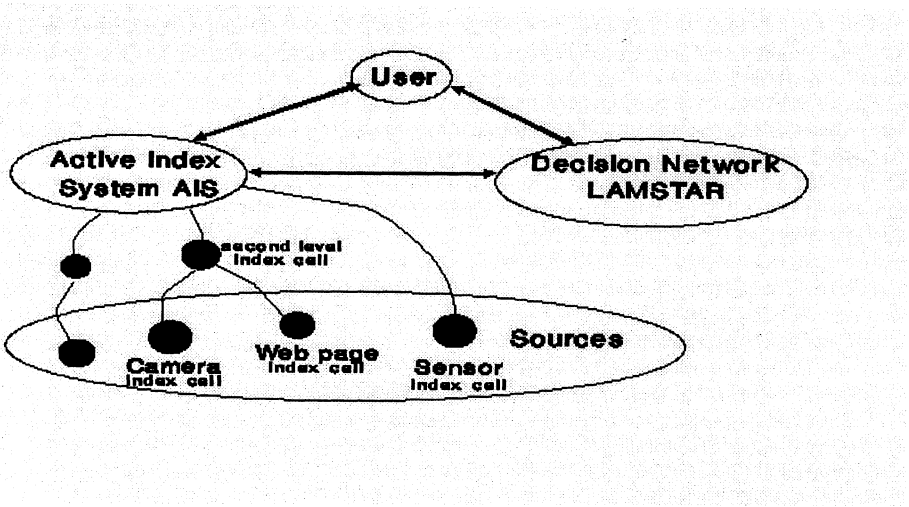

Figure 12. An active medical information system AMIS.

We will give an example to illustrate information fusion by horizontal/vertical reasoning. Patient information is abstracted from different media sources, including imaging devices, signal generators, instruments, etc. (vertical reasoning). Once abstracted and uniformly represented, the neural network is invoked to make a tentative diagnosis (horizontal reasoning). Using the active index, similar patient records are found by the Recursive Searcher (vertical reasoning). A retrieved patient record is compared with the target patient record (horizontal reasoning). If similar records lead to similar diagnosis then the results are consistent and the patient record (with diagnosis) is accepted and integrated into the knowledge base. If the diagnosis is different then the results are inconsistent and the negative feedback can also help the decision network learn.

In the vertical reasoning phase, in addition to comparing patient data, we can also compare images to determine whether we have found similar patient records. Therefore, content-based image similarity retrieval becomes a part of the vertical reasoning. Depending upon the application domain, image similarity can be based upon shape, color, volume or other attributes of an object, spatial relationship among objects, and so on.

This example illustrates the alternating application of horizontal reasoning (using the LAMSTAR neural network for making predictions) and vertical reasoning (using dynamically created active index for making associations). Combined, we have an active information system for medical information fusion and consistency checking.

A demo of the active medical information system can be found at: *http://www.cs.pitt.edu/~chang* and then click on Active Medical Information Systems.

MICE has also been used to prototype an emergency management system [Khali96], an intelligent multimedia information retrieval system [Catar98], a multimedia distance learning system [Chang98c] and other multimedia applications.

Chapter 9

Systems: The Design of Multimedia Languages

1. INTRODUCTION

The inherent complexity and size of many multimedia applications requires the introduction of proper software engineering techniques, languages, and tools for mastering the specification, the development process, and the dynamics characterizing their presentation. In Chapter 4, we described how visual languages can be extended in order to capture the dynamic behavior of multimedia objects [Chang96a].

The extended visual languages are called multidimensional languages and are still based on the concept of generalized icons [Chang91]. The user can access and animate multimedia information by composing multidimensional sentences, that is, by combining generalized icons according to some spatial and/or temporal relations. The extended visual languages can be used for the development of teleaction objects (TAOs) [ChangH95b], multimedia objects that automatically respond to events, and are particularly suitable for modeling multimedia presentations.

In this chapter we discuss the design of multidimensional languages for specific application domains. The chapter is structured as follows. In Sections 2 and 3 we describe in order the TAO model and the extended generalized icons. In Section 4 we review the concepts underlying a visual language design methodology. The new methodology is presented in Section 5, whereas an example is described in Section 6. Finally we discuss the methodology in Section 7.

2. THE TAO MODEL FOR MULTIMEDIA

TAOs are multimedia objects capable of automatically reacting to events and messages. The structure of the multimedia objects is represented through an hypergraph G, whereas the event driven dynamic structure is represented through a knowledge structure K called Active Index [Chang96a]. G is a graph G(N, L), where N is the set of nodes and L is the set of links. A node can represent a media type or even a TAO itself. Links can be of the following types: attachment, annotation, location, and synchronization. An example of TAO hypergraph is given in Figure 1.

The physical appearance of a TAO is described by a multidimensional sentence, which is a spatial/temporal composition of generalized icons [Chang91], [Chang96a]. A multidimensional language is a set of multidimensional sentences. The syntactic structure underlying a multidimensional sentence controls its dynamic multimedia presentation.

3. GENERALIZED ICONS AND ICON OPERATORS

Generalized icons are dual objects $x = (x_m, x_p)$, where x_m is the meaning and x_p is the physical appearance.

In visual languages the physical appearance x_p is an icon image. In multidimensional languages the concept of generalized icon has been extended to represent all the different types of media [Arndt97], [Chang96a]. The following types of generalized icons have been defined:

- Icon: (x_m, x_i) where x_i is an image
- Earcon: (x_m, x_e) where x_e is sound
- Micon: (x_m, x_s) where x_s is a sequence of icon images (motion icon)
- Ticon: (x_m, x_t) where x_t is text (ticon can be regarded as a subtype of 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 composite icon or multimedia sentence.

Also icon operators are dual objects $op = (op_m, op_i)$, The physical part (op_i) combines the physical parts of generalized icons, whereas the logical part (op_m) combines their meanings. Multidimensional sentences are constructed by combining generalized icons through earcon operators such as fade in or fade out, micon operators such as zoom in or zoom out, ticon operators such as text merge or text collate, and temporal operators [Allen91].

In TAOs generalized icons are represented by nodes in the hypergraph, whereas operators are represented by links. As an example, let us consider the multimedia presentation Salerno Multimediale, a CD-ROM describing

the city of Salerno. The presentation begins by displaying a cover image. After the user touches the screen, a background sound is played and animation starts. The latter is composed of a background image with a rotating label "Salerno Multimediale" on it. After few seconds the rotating label fades out and a falling curtain starts covering the background image. The animation yields another background image with a menu overlaid on it. The TAO hypergraph for this portion of the CD-ROM is shown in Figure 1.

4. A METHODOLOGY FOR VISUAL LANGUAGE DESIGN

Based on the concept of generalized icons, we have developed a methodology for the design of iconic languages [Chang94b], a subclass of visual languages. Successively, the methodology has been extended to accomplish the design of general visual languages [Poles98] and temporal visual languages [Chang97].

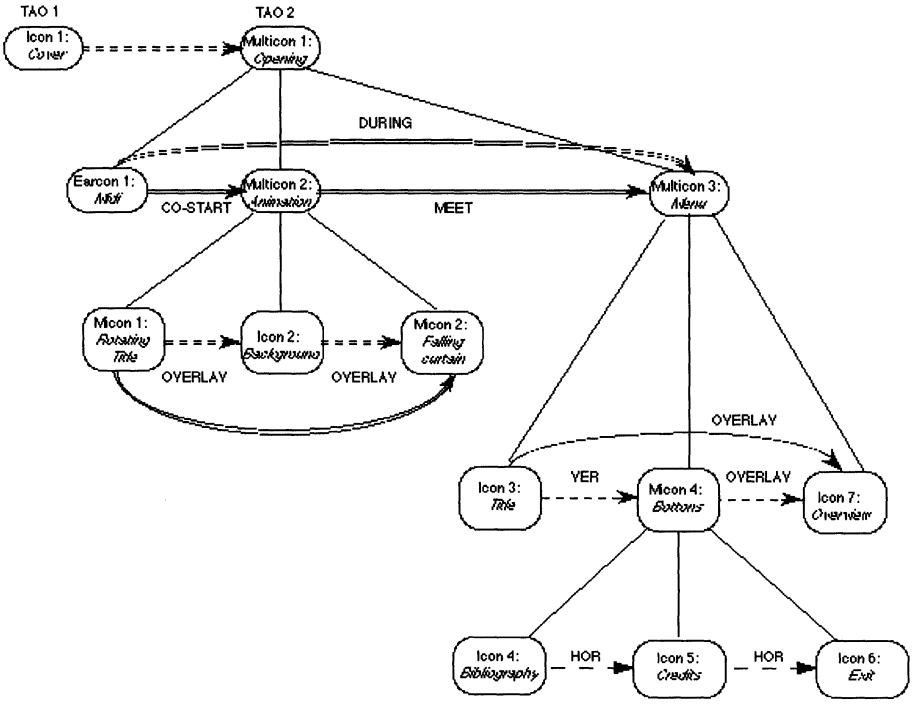

Figure 1. An example of the hypergraph structure.

The design problem for visual languages is to encode the elements of an application domain through visual sentences semantically close to them, according to a certain metaphor. Let K be the set of domain elements to be visually encoded, the phases to be executed in our design methodology are outlined in Figure 2.

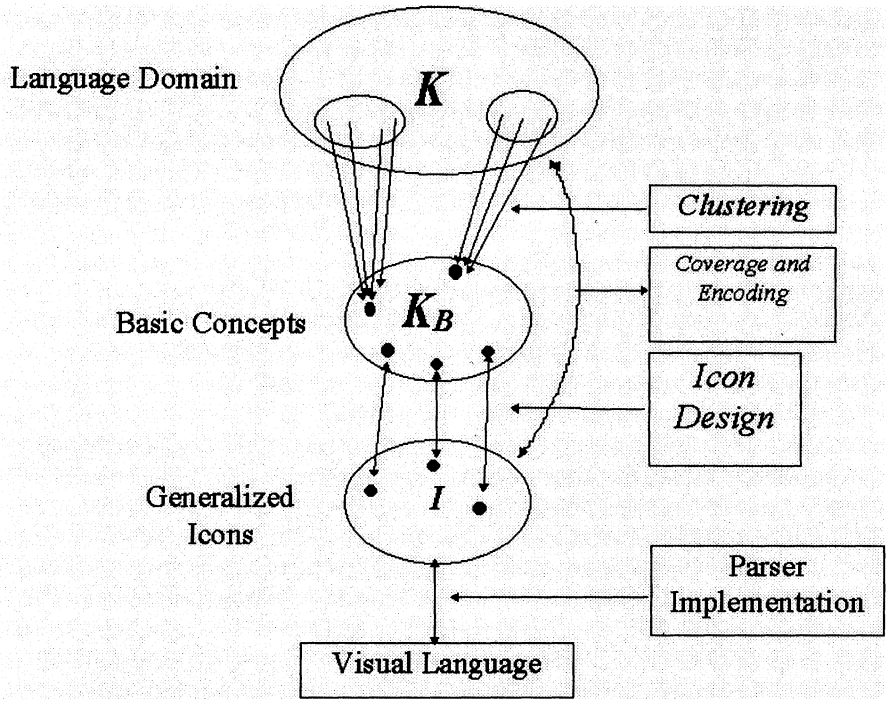

Figure 2. The methodology for visual language design.

Initially, we need to build the domain K and a reduced knowledge base to better characterize its elements. Then, we cluster K according to an adaptation of the K-B-means algorithm to obtain a reduced set KB of clusters representing basic concepts of the application domain. Successively, the phase of icon design is performed to sketch a visual representation for the elements of KB. For each word or concept $w_i \in$ KB we must sketch a visual representation for w_i to derive a new generalized icon $x = (x_m , x_i)$ such that xm includes w_i. At the end we obtain a set I of basic generalized icons. In the next step we should combine the icons in I through the operators of the icon algebra [Chang91] to form visual sentences, for the sake of visually encoding the whole set K. At this point, we should encode each element w i from the language domain through a visual sentence made of icons and operators. During the Coverage and Encoding phase we construct a visual sentence $S = (S_m , S_i)$, run inferences to derive its meaning

part S_m from the meaning parts of its component icons and icon operators, and then use S_i to encode the domain element w_i similar in meaning to S_m .

The final set of visual sentences form the language icon dictionary. Each record of this dictionary contains a domain element, the visual sentence encoding it, and a formal rationale explaining the association between the word and the visual sentence. The final language is tested through special tools to verify usability for the intended users. Finally, we need to construct a visual grammar in order to generate a parser [Costa97a].

5. THE EXTENDED METHODOLOGY FOR MULTIDIMENSIONAL LANGUAGES

We have extended our design methodology for visual languages to allow the design of multidimensional languages. The design problem for multidimensional languages is to derive a multimedia representation for the elements of an application domain. In general, this process includes content selection, media allocation, and media realization [Weitz94]. We see this process as the derivation of a certain number of TAOs representing the elements of the application domain in multimedia presentations. The association between these elements and the TAO is also dynamically ruled by the knowledge structure associated to the TAO. Thus, we have defined a design process to derive the multidimensional language for expressing the TAOs for a given domain.

5.1 Domain and Knowledge Construction

In this phase we have to build the multidimensional language domain K. As opposed to visual language design here the language domain includes more types of elements, such as images, sounds, etc. The frame structure for the knowledge base includes some of the attributes used for the design of visual languages, such as the attributes sound, time, location, color, and shape. Some other attributes depend upon the application domain and are used to express content. Their values include not only text but also image and sound. As a consequence, these attributes can also have a special index to be used for similarity matching. In fact, the similarity function to be used for multidimensional languages needs to find similarity in sound or image, for which it can use well known indexing and approximate matching techniques developed for multimedia databases.

5.2 Modeling and Clustering

In this phase we first structure the domain elements by using object oriented modeling techniques and then we perform Clustering by using the class diagram and a special distance function. The distance function still compares attributes to determine the similarity, but it will be using more sophisticated and approximate matching techniques because of the presence of complex types of data. For example, a text mentioning the painting of Leonardo Monnalisa should be considered close and therefore clustered together with a figure showing the image of MonnaLisa, and with all the images having similar visual characteristics, such as colors, shapes, etc.

5.3 Generalized Icon Design

The input to this phase is the class diagram, the object diagrams and the clusters determined in the previous phase. We need to translate their information in terms of generalized icons. Thus, we first construct the physical appearance of generalized icons and then their logical part. We can decide to provide both a visual and a sound representation for a textual information according to a certain metaphor.

After have sketched generalized icons for elementary information we apply icon operators to compose multidimensional sentences. We exploit the relationships of the class diagram and the clusters to understand the appropriate operators to apply.

5.4 Approximate coverage and TAO generation

For each element of the domain we compute a set of multidimensional sentences and a score indicating the type of similarity with a rationale associated. This information will be used for TAO construction.

After have constructed a set of multidimensional sentences covering the language domain, we can build a visual grammar and semantic routines to produce the TAOs covering the domain elements. The parsing of the sentences will control the multimedia presentation.

6. CASE STUDY

In this section we show an example on the use of our methodology for the development of a multidimensional language to transform lectures into multimedia presentation formats. We started from a domain language made of textual lectures with transparencies comprising text, figures, tables, and

movies on specific subjects. The goal was to derive the multidimensional sentences expressing the TAOs for the multimedia presentation of the lectures. An abstract class diagram for this example is shown in Figure 3.

Let us consider a transparency from a medical lecture on meniscal surgeries as shown in Figure 4. The text item Umbrella handle has associated the following frame in the knowledge base:

SLOT: VALUE
NAME: Umbrella handle
SHAPE: Sketch(Umbrella handle)
LOCATION: Middle of Knee
IMAGE: Overlay (This.Shape, Knee CT Scan)
TIME: Before(Before(Co start(This.text, This.Image), "Circle shape"), Movie1)

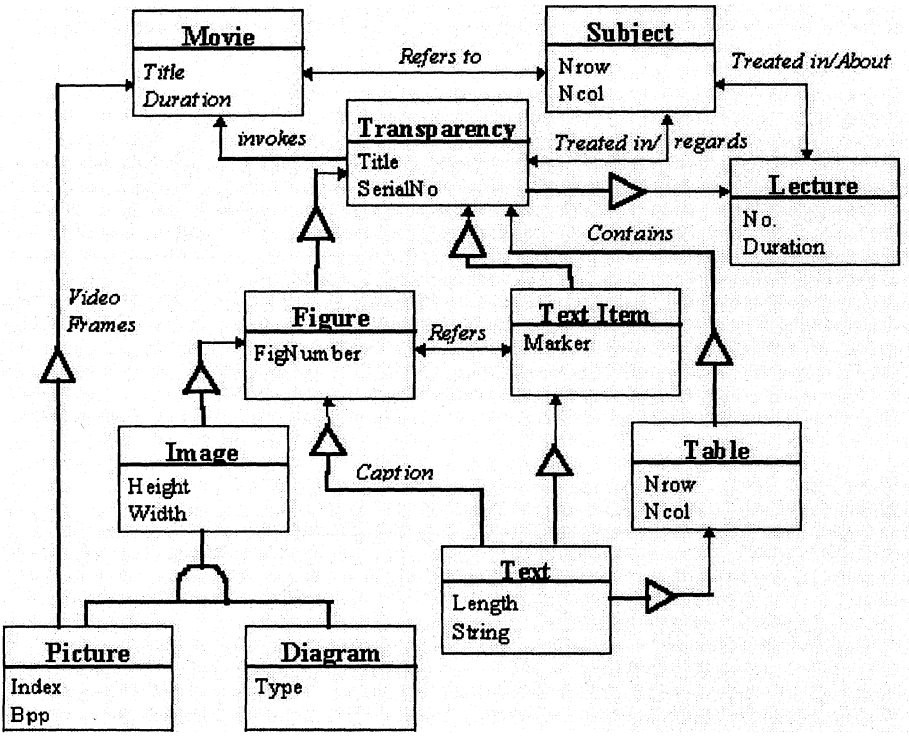

Figure 3. The class diagram for the Lecture example.

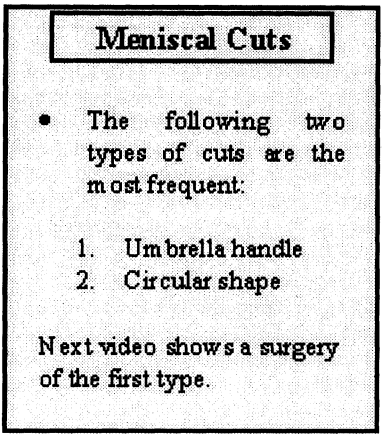

Figure 4. A transparency on Meniscal Surgeries.

In the object diagram there will be an instance Meniscal Cuts of the class Transparency, which is connected to the instance Meniscal Surgery of the class Subject through the relationship Regards, to the instance Movie1 of the class Movie through the relationship Invokes, and is composed of three instances of the class Text Item.

The Image attribute is a query to an image database [Nappi98]. We run such queries on the medical images of the language domain by using the system FIRST [Nappi98] during the clustering phase. In this way we could cluster images with similar meniscal anomalies together. During the Generalized Icon Design we produced a vicon Movie1 and a ticon for each of the three text items in the transparency. Then, the *part-of* relationship between the transparency and its three text items suggested the introduction of a multicon for the whole transparency Meniscal Cuts and the application of the attach operator for each of the three text items. We applied the spatial operator ver to combine the ticons. The Invokes relationship and the TIME attribute from the frame associated to the transparency suggested the application of the temporal operator before to combine the multicon with the vicon. The SHAPE attribute of the the textual items "Umbrella handle" and "Circular shape" suggested the sketch of two icons each depicting one of the two shapes. The sketch queries contained in the IMAGE attribute caused the linking during the clustering phase to examples of CT scan images reporting similar knee anomalies. We could then decide to enrich the presentation by combining the two ticons with either the two icons or even with some of the CT scan images (represented as icons) resulting from the queries. A further

decision was to be made on the spatial and temporal operators to use for combining ticons and icons. For example, each ticon could be combined with the associated icon by using the spatial operator overlay and the temporal operators co start, co end.

The entry Meniscal Cuts in the language icon dictionary will have associated several candidate multidimensional sentences and the rationale for each of them.

An example of a rationale follows:

[Multicon[Meniscal_Cuts] *attach*
[[ticon[umbrella] *overlay* + *co_start* + *co_end* icon[umbrella_cut]]
vertical [ticon[circular] *overlay* + *co_start* + *co_end* icon[circular_cut]]]
before Vicon[Movie1]

We can easily generate the associated TAO from this rationale.

7. DISCUSSION

We have presented a methodology for the design of multidimensional languages. The methodology serves as a prescriptive model for designing multidimensional sentences to be used for visually specifying the structure of Teleaction Objects. We need to further experiment the proposed methodology on a broader class of multimedia applications. Moreover, we need to refine logical icon operators to increase the generation of knowledge for the active index of TAOs.

Chapter 10

Systems: Distributed Multimedia Systems Design

One of the challenges in the design of a distributed multimedia system is to devise suitable specification models for various schemas in different levels of the system. Another important research issue is the integration and synchronization of heterogeneous multimedia objects. In this chapter, we present our models for different multimedia schemas and transformation algorithms that transform high-level multimedia objects into schemas that can be used to support the presentation and communication of the multimedia objects.

1. INTRODUCTION

Recent advances in high speed communication networks, mass storage, digital video, data compression, as well as the advocacy of the Information Superhighway by the government, have stimulated the research and development of the distributed multimedia system (DMS).

One of the challenges in the design of a distributed multimedia system is to devise a suitable specification model for various schemas in different levels of the system. Another important research issue is the integration and synchronization of heterogeneous multimedia objects (MMOs).

Related works regarding the timing specification, i.e., the synchronization, of multimedia objects for the purpose of presentation can be found in [Bulte91, Hardm94, Li94, Little93a, Little90b, Stein90]. The synchronization and integration of multimedia objects in the communication network layer of multimedia systems has also been discussed in [Little90b, Little91, Little93a, Little93b, Stein90].

In order to satisfy the requirements, for example real-time delivery, of some multimedia applications, the quality of service (QOS) support of the communication network and of the operating systems has been an important research topic [Campb94, Little91, Merce93, Ramae92, Son93]. The quality of service is also considered in the presentation layer of the multimedia systems [Fujik95, Staeh94].

In our view, there are three different multimedia schemas in the DMS. The Multimedia Static Schema (MSS) specifies the static structure of the composite multimedia objects, including the temporal and spatial relations among the objects.

The Multimedia Data Schema (MDS) specifies the properties of MMOs, the properties of a set of objects as a whole, and the temporal relations between the objects, thus supporting the integration and synchronization of MMOs

The Multimedia Communications Schema (MCS) is derived from MDS by adding further communication control and synchronization requirements in order to satisfy the capacity constraint of processors, bandwidth constraint of the communication network, as well as to effectively utilize the service of the communication network. Finally, the network primitives are derived from MCS to perform the transmission of MMOs [Lin94, Lin95].

A hypergraph model is suitable to specify the structure of composite MMOs [Hou94] and is employed to specify the MSS in this chapter. The G-Net model [Deng90, Deng91], a Petri net [Reisi85] based model, is intended for the design and specification of complex information systems [Chang92]. Petri net based models have been proposed to serve as a suitable model for the unified framework to specify different levels of the DMSs [Little90b, Znati93] and the G-Net model is adopted in this chapter as the model for both MDS and MCS. Thus, the transformation from an MDS to its corresponding MCS can be formulated as the G-Net transformation from the G-Net description of MDS to that of MCS [Lin94, Lin95]. The special hierarchical feature of the G-Net [Deng90, Deng91] is utilized in the transformation.

A key module in a distributed multimedia system is the Object Exchange Manager (OEM). In order to exchange multimedia objects in a distributed system, a uniform representation, such as MHEG (Multimedia and Hypermedia Information Object Expert Group) [Colai94], is needed to maintain all information of an MMO. The Object Exchange Manager maintains and manages the uniform representation and interacts with other system modules.

Our approach is neither to design a communication network protocol to support QOS, nor to develop a powerful multimedia authoring tool. What we propose is a novel idea of the feasibility of the transformation between

multimedia schemas, and the transformation will provide a unified framework to support a link from application layer, synchronization and integration in the presentation layer, down to the communication network layer.

The chapter is organized as follows. In Section 2, the architecture of a DMS and the transformation approach are presented. The models to specify multimedia schemas and the transformation algorithms are described in Section 3. An example that illustrates the transformation algorithms is presented in Section 4. Section 5 describes the transformation from the MCS to the corresponding network primitives that will accomplish the transmission of the MMOs. The design and implementation of the OEM is described in Section 6. Section 7 describes the implementation of our experimental multimedia system, based on the transformation approach and the object exchange manager. Section 8 discusses the future research issues.

2. THE ARCHITECTURE AND THE TRANSFORMATION APPROACH

The architecture of a distributed multimedia system can be divided into three layers: the application layer, the system layer, and the communication layer. Figure 1 depicts the three layers and the relation among the layers. When a user invokes the application, such as a multimedia message/mail system, and initiates an MMO instance, an MSS is created, for example, by a multimedia editor. The MSS is then transformed into the corresponding MDS to support the presentation of MMOs and to serve as the input for transformation in the next phase. Subsequently, the transformation rules are applied to transform the MDS to its corresponding MCS, and then to the network primitives to conduct the transmission of the MMOs according to the specification [Lin94, Lin95].

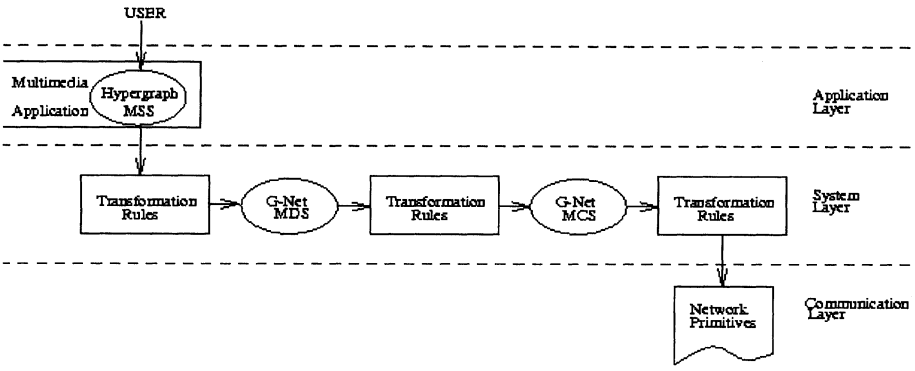

Figure 1. Transformation between multimedia schemas in a distributed multimedia system.

Due to the heterogeneous characteristics of MMOs [Stein90] and various platforms which invoke multimedia applications, the specification of the transmission of MMOs involves the following factors:

1. the attributes of MMOs, including
   - the size of each MMO
   - the type of each MMO
   - the structure of MMOs
2. the hardware specification that describes
   - the type of the machine
   - the capability to display graphic objects
   - the capability to display image objects
   - the capability to playback audio objects
   - the capability to playback video objects
   - the type of display attached to the machine
   - the type of audio system of the machine
   - the capacity of the hard drives of the machine
   - the bit rate of the communication network
   - the quality of service (QOS) [Campb94, Fujik95, Ramae92] of the communication network and the overall system. A request for transmission can be accepted only when the transmission can be guaranteed under the current specification of the QOS.
   - the IP address of the user.
3. the transmission scenario specified by the user, such as
   - which objects in the MMOs are to be transmitted
   - the timing requirement
   - the quality of service requested
   - the heuristics to be invoked in order to negotiate with the QOS manager. For example, in case the network cannot guarantee the transmission due to a bandwidth constraint, progressive

transmission [Tzou87] for objects having types *image* or *video* might be employed to transmit a low-resolution image/video first, with a smaller size in comparison with the original object, and then the refined image/video progressively.

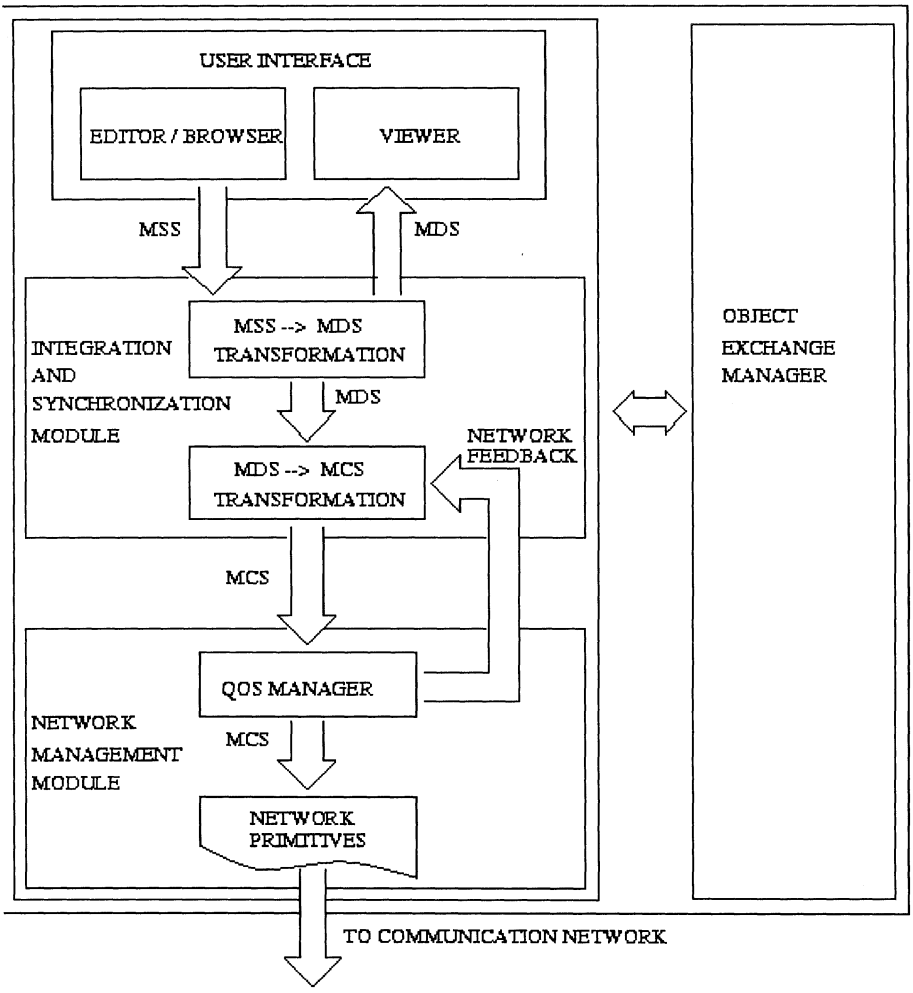

Figure 2. The architecture of a distributed multimedia system.

The block diagram of the architecture of a distributed multimedia system and the transformations between multimedia schemas in the system is depicted in Figure 2. This architecture features a unified approach for the modeling of multimedia application, presentation and communication, as a structure consisting of the G-Net specification, attributes, procedures and relations. The system consists of the following modules: User Interface, Object Exchange Manager, Integration and Synchronization, and Network

Management.

2.1 User Interface

User Interface module deals with the domain-specific applications of DMS. Possible applications could be: delayed teleconferencing [Hou94] for the virtual office, virtual library and virtual laboratory. User Interface module is closely related to applications. There are two kinds of user interfaces: Generic user interface and application specific user interface. Generic user interface is application independent, serving as a user interface with the kernel DMS. Application users may not have access to this interface if an extra application specific user interface and application layer are built upon it. Four basic functions are provided in the generic user interface: MMO editing, MMO browsing, knowledge generation, and MMO presentation.

MMO editor provides facilities for editing an MMO hypergraph structure. By providing a set of drawing functions, it enables users to create, add, and delete nodes or links in the MMO hypergraph structure. It is a user interface that gets the MMO hypergraph structure from the user. The obtained hypergraph structure is stored in an internal C structure and can then be translated into the Object Exchange Format (OEF) in ASCII (see Appendix C).

MMO browser lets users walk through the MMO hypergraph structure created by MMO editor. If one is interested in some node or link, he/she simply selects(clicks on) that node or link. All attributes of that node or link are then displayed. Generator is a knowledge acquisition facility. By providing users with a user interface, it gets knowledge associated with an MMO instance and converts it into a script of rule set. Knowledge is represented as a (condition, event, action) triple internally. But users need not be bothered to understand this internal representation. The presentation module presents (displays or plays) the MMO by following the specification in MDS and accessing the MMO hypergraph structure. The MDS and the hypergraph structure of the MMO, i.e., MSS, are obtained from the Object Exchange Format, which in turn may come from a remote site via network.

Object Exchange Manager

Object Exchange Manager(OEM) maintains an Object Exchange Format(OEF) which contains all information of an MMO. It is a key module in the DMS system which interacts with all other modules of the system. OEM will be discussed in depth in Section 6.

2.2 Integration and Synchronization

Integration and Synchronization module looks into the overall properties and behaviors of an MMO. It deals with spatial as well as temporal relations within an MMO. It also takes into account the communication and synchronization requirements. It performs MSS to MDS and MDS to MCS transformations.

2.3 Network Management

Network Management module consists of a set of network primitives carrying out MMO transmission. The supported network primitives are: connection establishment, connection closing, message sending, message receiving, and synchronization. It provides the network service as the result of negotiation with the underlying transport service provider.

As shown in Figure 2, a user can invoke a Multimedia Editor/Browser to compose MMOs. After the composition, MSS of the composed objects is generated and sent to the Integration and Synchronization Module for transformations into MDS and MCS. The Multimedia Viewer can then utilize the MDS to view the presentation of the MMOs just created. If the QOS manager cannot guarantee the transmission based on the MCS, the Integration and Synchronization Module has to modify the MCS to negotiate with the QOS manager. After the negotiation is achieved, the MCS will be transformed into network primitives to perform the transmission. By employing the concept of object exchange [Colai94], the Object Exchange Manager is devised to maintain and manage the Object Exchange Format [Xiang95] and to interact with the other modules in the system.

3. MULTIMEDIA SCHEMA MODELS AND TRANSFORMATION ALGORITHMS

3.1 Multimedia Schema Models

An Object Exchange Format (OEF) has been defined to specify MSS [Xiang95]. The underlying structure is a hypergraph structure. In the hypergraph structure, each node represents an MMO, either a composite object or a basic object. Among the nodes, there are a set of hyper-links which connect nodes. Five types of links are described as follows:

- *attachment links* represent the structure of MMOs. An attachment link connects a composite object with one of its sub-objects. Therefore, we can have a hierarchical view of the whole MMOs.
- *reference links* link to the references of objects.
- *spatial links* specify spatial relations among objects.
- *temporal links* specify temporal relations among objects.
- *annotation links* connect the annotating and annotated objects.

On the other hand, the G-Net model is adopted to specify both MDS and MCS of a distributed multimedia system [Lin94, Lin95]. A G-Net consists of two components which are the Generic Switch Place (GSP) and the Internal Structure (IS) [Deng90, Deng91]. The GSP provides an abstraction of the IS, while the IS, a modified Petri net, contains mainly place primitives and transitions and describes the actual specification of the G-Net.

The G-Net descriptions of the MDS and the MCS can be specified as follows:

1. Specify the global information about the schema in the GSP of the G-Net.
2. Use place primitives to specify the information about each individual object, synchronization points, and delays.
3. Use transitions to specify the temporal relations among the objects. However, if the temporal relations are specified in the places, the transitions can fire instantaneously and the model has the advantage of compactness of representation [Little90b]. The G-Net model adopts the latter specification.

3.2 The Transformation from MSS to MDS

As the hypergraph structure representing MSS in the OEF could be complicated due to its rich set of links, the MSS is transformed into an intermediate model first then into the MDS in order to simplify the transformation. The intermediate model adopted here is based on the CWI Multimedia Interchange Format (CMIF) model [Bulte91, Hardm94] augmented by adding a delay attribute to each node in it. The CMIF model is a hierarchically structured model for representing and manipulating multimedia documents. In the tree view of the hierarchy of the CMIF model, each internal node specifies the temporal relation, either parallel or serial, among its children nodes and each leaf node represents a multimedia object. By adding a delay into each node, various temporal relations can be represented in a unified schema as described in the following paragraph. The tree structure allows the employment of recursive algorithms to perform the

transformation. However, there is a restriction of the CMIF model that it only provides synchronization between sibling nodes, i.e., nodes of the same parent node. Thus, the temporal links between non-sibling nodes are not processed in the transformation from MSS to IM and stored in a set of links.

The temporal relations among objects are defined by the links in the MSS. In the construction of the intermediate model, a temporal node is generated in the IM for sibling nodes connected by a temporal link in the MSS. This temporal node will become a child of the node in the IM corresponding to the node to which those sibling nodes are attached in the MSS. There are two categories of temporal relations between two objects: parallel and sequential. The sequential category consists of two temporal relations which are *before* and *meet*, while the parallel category consists of five temporal relations: *co-begin, co-end, equal, overlap*, and *during* [Little90b].

The synchronization attributes of a temporal link include the temporal relations and a temporal value, e.g., the gap between the end of an object and the beginning of the other in relation *before*. Based on the synchronization attributes, a delay attribute can be added to each node connected by the link to specify the delay, where, of that node. A special value d, which stands for *don't care* for delay , is used for the temporal relation not specified definitely among the objects. Three assumptions are made here. One is that the temporal relation between sibling nodes in the MSS not linked by a temporal link is defined as parallel with as d. Another is that the temporal relation among the annotating objects and annotated object is the parallel relation *during*, which is a reasonable interpretation. The third is that each object linked by a reference link should have its own MDS, and consequently MCS, as it will be presented and/or transmitted only when requested. Therefore, the referenced objects will not be included in the MDS transformed from the MSS. However, the information is kept in the MSS and can be brought up when the objects are referred.

Figure 3 illustrates the construction of an intermediate model.

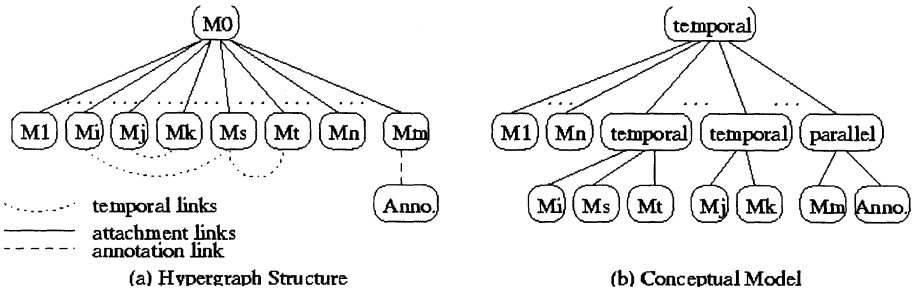

Figure 3. Construction of the Intermediate Model.

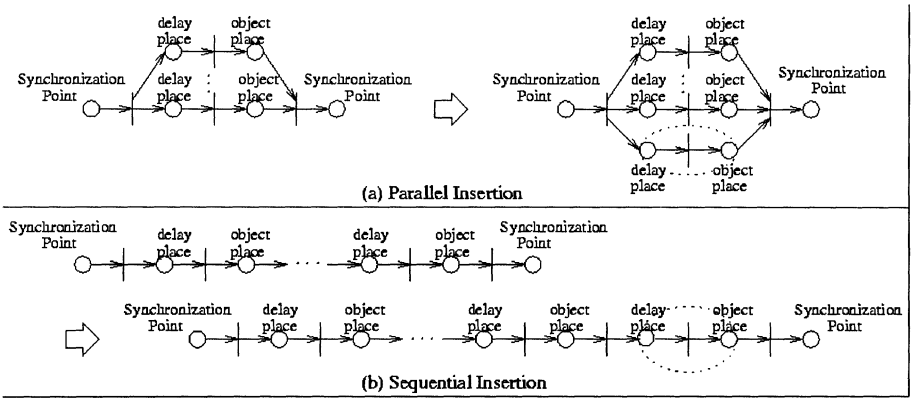

Figure 4. The insertion of an object in the IM-to-MDS' transformation. The object is enclosed in a dotted ellipse.

The transformation from IM to MDS is divided into two phases. In the first phase, a temporary G-Net MDS' is transformed recursively from the intermediate model. In the second phase, the temporal links stored in the set L are processed by adding auxiliary places and transitions onto MDS' to complete the construction of the MDS.

When transforming from IM to MDS', two places and one transition are created to represent each node in the corresponding MDS'. One place is the input place while the other place is the output place of the transition. The input place, which is called the *delay place*, represents the delay of the node, handles temporal links linking non-sibling nodes, and provides possibly further synchronization. One type of scenario for more flexible synchronization is *fuzzy scenarios* [Li94]. For example, three multimedia objects are specified to be presented in parallel while the order for presentation is not that important. With the employment of the value d, we can formulate a G-Net that describes this scenario. The modeling of fuzzy scenarios and other synchronization scenarios illustrates a unified

construction of the MDS' can be achieved. The output place indicating the object represented by the node and is called the *object place*. There are two scenarios of inserting an object into the schema according to the temporal category of its parent node and each scenario should be handled differently. Figure 4 describes both scenarios of parallel insertion and sequential insertion.

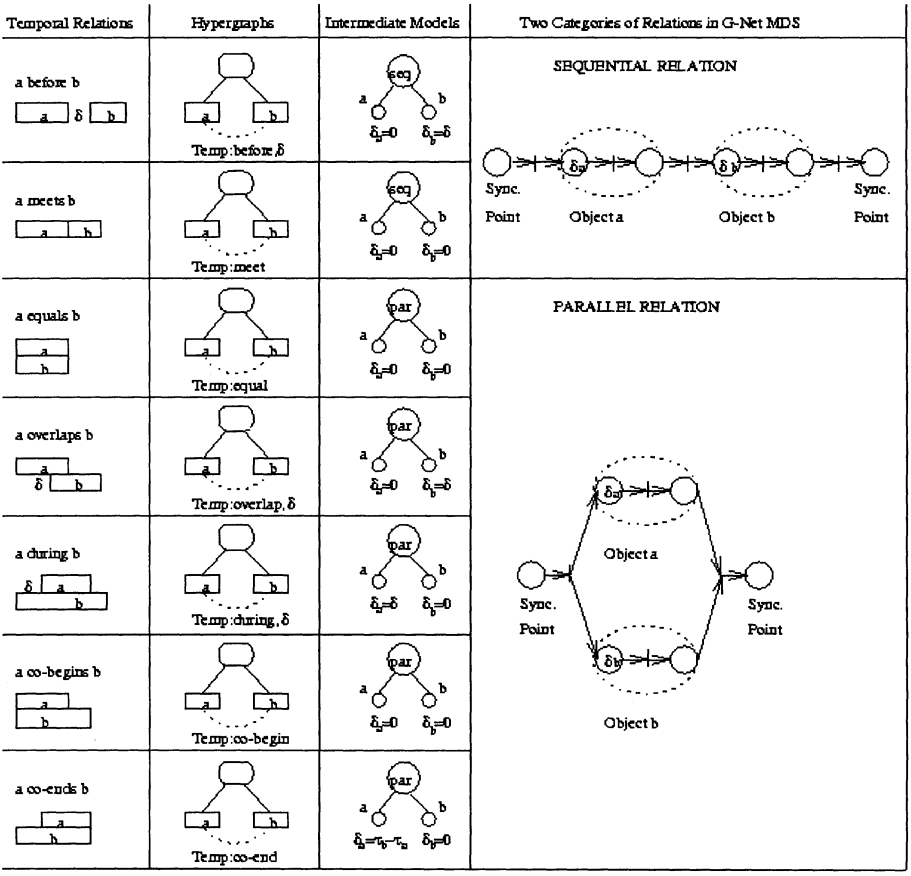

Figure 5. The temporal relation between two objects and the corresponding structures.

The temporal relations between two objects and the corresponding hypergraph structure, intermediate model, and MDS' that represent the relations are depicted in Figure 5.

The last step in the transformation is to process the non-sibling links stored in the set L previously. The idea is to make use of the delay place of each object as well as to generate auxiliary places and transitions to provide synchronization according to the temporal attribute of a link.

Figure 6 shows the detail of processing a link from object M1 to M2.

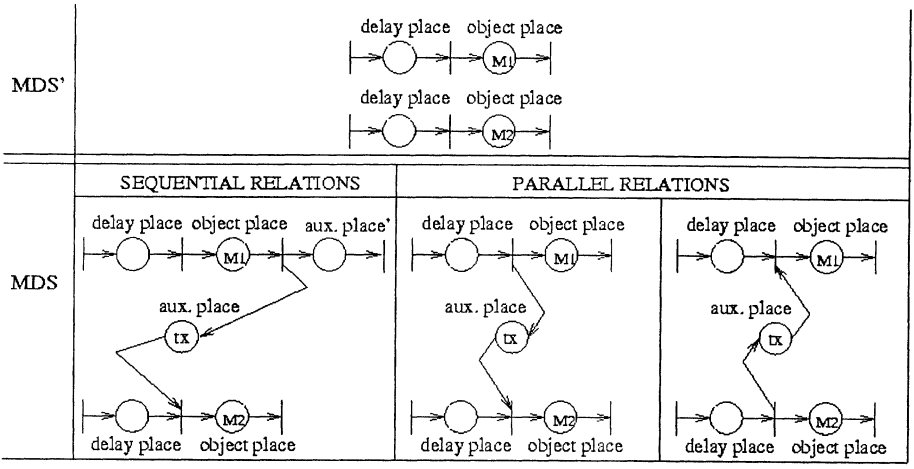

Figure 6. The construction of MDS from MDS' by processing non-sibling links in the MSS

The value of the duration attribute of the auxiliary place is shown by the table in Figure 7.

Temporal attribute of link S $S.attribute$	$P_{aux}.duration\ t_x$
Meet	0
Before,δ	δ
Equal	0
Co-begin	0
Overlap,δ	δ
During,δ	δ
Co-end	$\lvert\tau_a - \tau_b\rvert$

Figure 7. The value of the duration attribute of the auxiliary place.

The high-level description of the MSS-to-MDS transformation algorithm is illustrated in Algorithm 1. The details of the subroutines are given in Appendix A.

```
Procedure MSS-to-MDS (OEF_MSS, G_MDS)
begin
  /* Transform the MSS to an intermediate model of tree structure IM.
     Links in the MSS are processed except non-sibling temporal links
     in the MSS which will be stored in the set of links L.  */
  MSS-to-IM (OEF_MSS, IM, L)
  /* Call recursive function to construct a temporary G-net G_MDS from IM
  . */
  R = root of IM
  newplace(P)
  G_MDS' = IM-to-MDS' (R, P)
  /* Process non-sibling temporal links to complete the construction of
     G-net G_MDS. */
  MDS'-to-MDS (G_MDS', L, G_MDS)
end
```

3.3 The Transformation from MDS to MCS

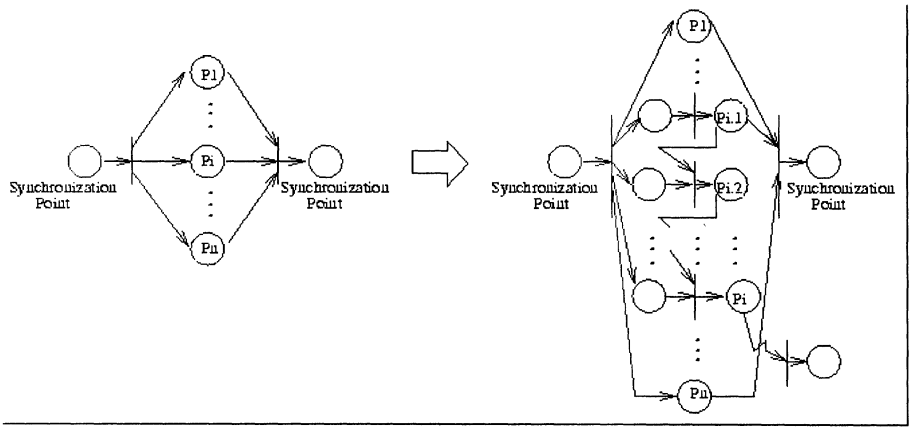

Figure 8. G-Net transformation when applying progressive transmission on object.

Once the MDS is constructed, we do not have to reconstruct the corresponding MCS from scratch. By defining a transmission vector that specifies which types of objects are to be sent and the parameters of the heuristics, for example, levels of progressive transmission, the MCS can be constructed from MDS by applying the delete operation and heuristics such as progress operation on the corresponding objects.

The transmission vector serves as an input parameter of the transformation. The values of the entries in the transmission vector are defined by both the hardware capacity of the machines and transmission specification provided by the user.

Another input parameter is the quality of service (QOS) of the communication network which consists of two fields, namely *QOS.request* and *QOS.support*. *QOS.request* is the QOS requested by the user while the *QOS.suppor* is the QOS guaranteed by the system. Therefore, given the MDS, the transmission vector, and QOS, the transformation is performed in three steps described as follows.

1. Copy G_{MDS} to G_{MCS}.
2. Check the transmission vector to delete the places in the G-Net structure of the *MDS* corresponding to the objects not to be sent. In order to keep the semantics of the schema, the place is logically deleted by re-defining the place, such as making it a synchronization point, without physically deleting the place from the net.
3. If *QOS.support* is less than *QOS.request*, apply heuristics to degrade *QOS.request*. Replace each place corresponding to an object whose quality needs to be degraded with an *ISP* [Deng90, Deng91] containing a G-net representing the heuristics invoked. For examples, a G-Net representing progressive transmission can be applied to image objects, and a G-net representing *dropping frames* can be applied to video objects. In the case where progressive transmission is employed for an image the algorithm progress (short for progressive) will apply. Figure 8 shows the transformation of this scenario wherein the algorithm progress is applied on object and a rough object is sent first, followed by its progressive details. However, after an object is sent, the transmission of other objects does not have to wait for the transmission of the details to be completed. The details could be transmitted later when the bandwidth of the communication network is enough.

The following algorithm describes the transformation from MDS to MCS.

Procedure MDS-to-MCS(G_{MDS}, *TV*, *QOS*, G_{MCS})
/* Algorithm 2: G-Net Transformation from MDS to MCS */
begin
$G_{MCS} = G_{MDS}$
for each object place *P* in G_{MCS}

```
    if (TV[MMType(P.object)] == NO)
    Delete(G_MCS, P.object)
    if (QOS.request > QOS.support)
      ratio = QOS.request / QOS.support
      for each object place P in G_MCS
        switch (Heuristics)
          COMPRESS: G_p = ISP(h1(ratio))
                break
          PROGRESS: G_p = ISP(h2(ratio))
                break

        replace(P, G_p)
end
```

For example, we can define *QOS.request* and *QOS.support* as follows:

$$QOS.request = \psi_1(G_{MDS}, T)$$
$$QOS.support = \psi_2(QOS.network)$$

where G_{MDS} is the MDS, T is the time specification, and *QOS.network* is the QOS parameters provided by the communication network.

For example, a full-colored (24 *bits/pixel*) image of dimension 320 x 240 is requested to be transmitted in over a communication network with bandwidth 14400 *bps*. The size of the image is 1800 *Kbits* and the bit rate requested is 225 *Kbps* > 14400 *bps*. The utilization ratio *ratio* = 16. Therefore, we need to apply heuristics to reduce the quality of the image in order to fit in the service provided. Two possible heuristics, namely image compression and progressive transmission, are depicted in Figure 9.

In general, *QOS.request* and *QOS.support* are also vectors, so that the component-wise comparison will lead to the execution of different heuristics.

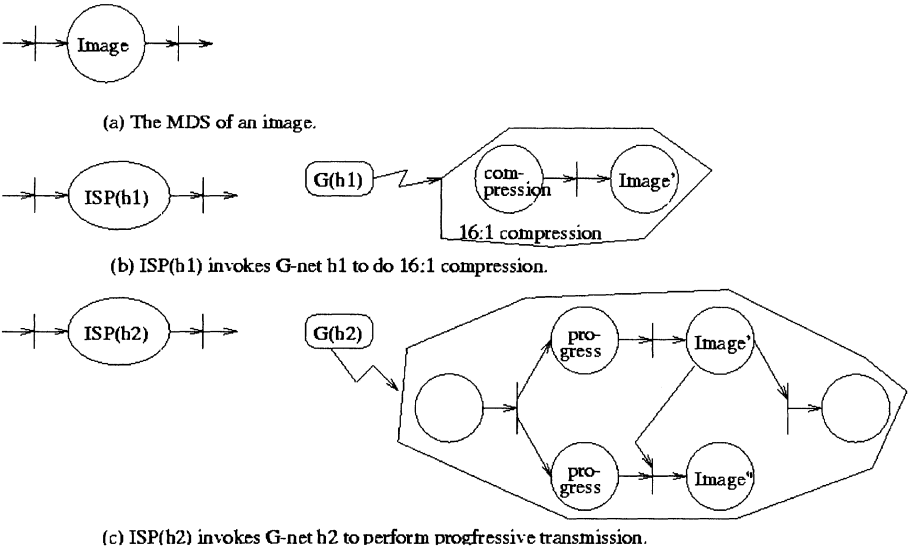

Figure 9. Applying heuristics to the image to be transmitted.

4. AN EXAMPLE OF THE TRANSFORMATIONS

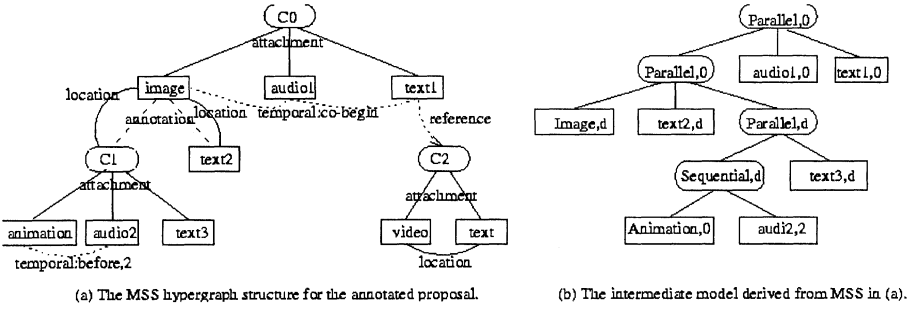

Figure 10. The MSS hypergraph structure and the intermediate model of the annotated 29medical report.

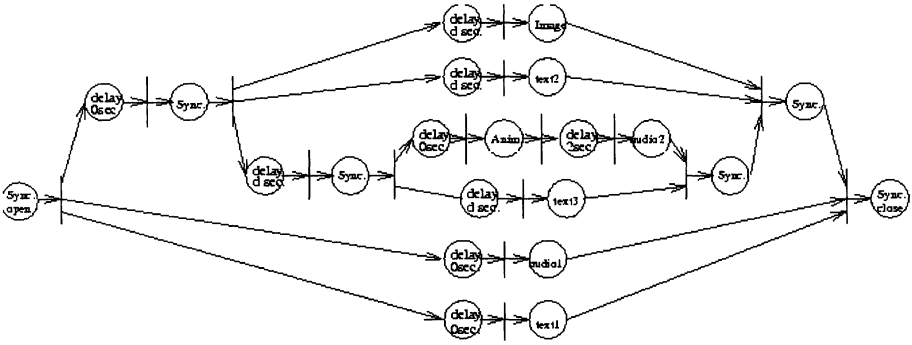

Figure 11. MDS derived from MSS shown in Figure 10.

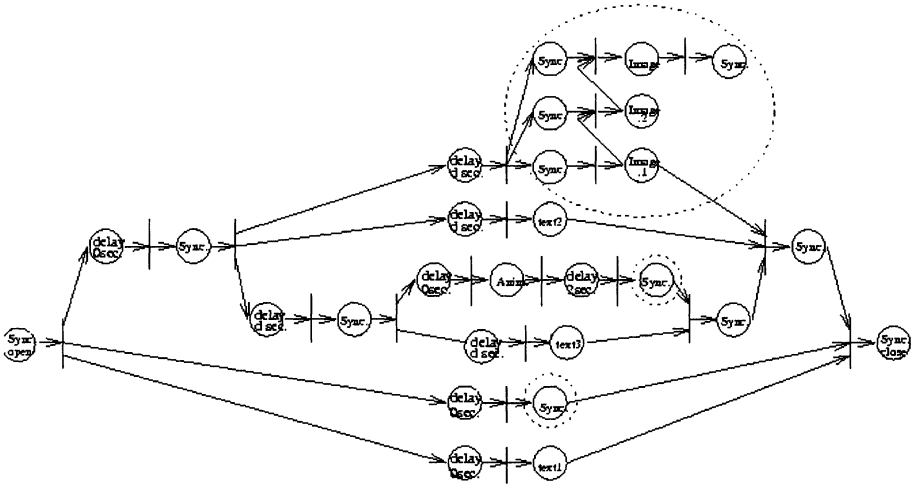

Figure 12. MCS derived from MDS shown in Figure 10. The places enclosed by dotted line represent objects processed in the transformation.

A medical doctor Smith is preparing a report for his supervisor Kessler and his group members Wang and Larson. His report *C*0 contains a text file of a medical record, an audio memo recorded by himself, and a nuclear image of the patient. The text also refers to a confidential report *C2* for his supervisor Kessler only. Suppose Smith receives two annotations about the image in his proposal. One annotation is a text while the other annotation *C*1 is composed of an animation, an audio, and another text where the audio will start to play two seconds after the animation has been played. Figure 10a shows the MSS of report *C*0 proposal which includes all the different

types of nodes and links in the hypergraph structure to specify the static structure of the multimedia objects.

The MSS modeled by the hypergraph structure can be transformed into the corresponding MDS according to the MSS-to-MDS algorithm described in Section 3.2. Figure 10b illustrates the intermediate model derived from the MSS. The number after the comma in each node represents the value of the delay attribute of the object. Figure 11 is the MDS constructed from the intermediate model.

Suppose that the receiving end does not have audio device and the bandwidth of the communication network is not enough to transmit the image object unless the size of the image is reduced in order to obtain the guarantee form the QOS manager. By applying Algorithm 2, MCS is transformed from MDS by deleting the audio objects and then employing progressive transmission on the image object. The resulting MCS is shown in Figure 12 in which a two-level progressive transmission is applied on the image object, where image.1 is an image of a lower resolution to be transmitted first while image.2 and image are the data to be transmitted progressively in order to fill up the detail of the original nuclear image.

Optimization is possible to reduce the complexity of the G-Net of an MDS. Various heuristics can be devised based on properties of the application. Figure 13 shows an optimized version of MDS in this example by deleting places with delay 0 and their output transitions.

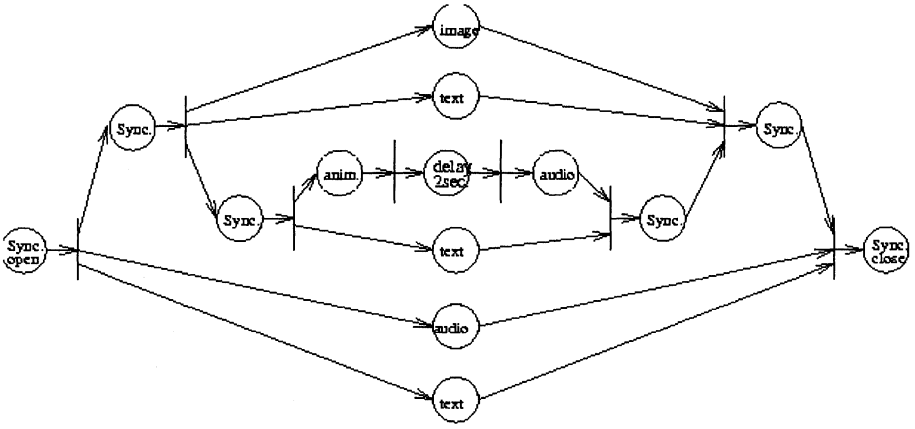

Figure 13. Optimized version of MDS shown in Figure 10.

5. THE TRANSFORMATION FROM MCS TO NETWORK PRIMITIVES

The MCS is depicted by a G-net graph. A G-net is composed of places and transitions. In a place we can describe what kind of action will be performed. For example, a commonly performed action is to send an MMO into the multimedia communication network. A transition tells us which place(s) will be enabled next.

There are five network primitives [Lin94, Lin95]:

- open(MMO_id)
- send(MMO_id, type, time_spec, priority, progress_flag)
- retrieve(MMO_id, type, time_spec, priority, progress_flag)
- synchronize(MMO_id)
- close(MMO_id)

The MCS will be transformed into a sequence of these five kinds of network primitives. Using for example TCP/IP Unix mail commands, MMOs can be sent or retrieved between two machines.

All MMOs can be classified into three classes, which are real time, restricted, and best effort. The real time class includes the objects specified as real time by the users, the objects to be retrieved and specified as real time retrieval by the remote users, and the first object in a progressive transmission. The restricted class includes all objects specified by the users as time restricted objects. For example, a user can request the transmission of an image to be completed in eight seconds. The best effort class includes all objects with no time restriction, for example, the details of a progressively transmitted object.

The program MM_Send will send all MMOs in the real time class first, then the restricted class, and the best effort class last. When MM_Send sends objects in the restricted class, it must make sure that all MMOs in the real time class have already been sent. In the same way, when MM_Send sends objects in the best effort class, all MMOs in the real time and the restricted classes must have been sent beforehand. A special type of object, retrieve, can be sent to request retrieval of MMOs from the receiver. A G-Net of MCS, , is embedded in the retrieve type object to specify the communication schema of the retrieval.

Another program MM_Receive will check either the system mailbox or a directory specified by the user for the retrieval of MMOs. In the former case, the MMOs are extracted from the system mailbox and put into directory /tmp/MM ready to be transmitted. Another task of is to deal with

retrieve type of MMOs and informs MM_Send program to send MMOs to the remote requester according to the embedded $MCS_{retrieve}$.

6. MULTIMEDIA OBJECT EXCHANGE MANAGER

6.1 Design and Implementation of OEM

To exchange the MMOs in the distributed multimedia system, we need a uniform representation which is able to maintain all information of an MMO. This requirement leads to the formulation of an Object Exchange Format (OEF) (see Appendix II). The OEM maintains and manages the OEF and interacts with other system modules. The object oriented approach is adopted in designing the OEM.

6.1.1 The Object Exchange Format

The OEF consists of five parts: header part, node part, link part, knowledge part, and raw data part, as illustrated in Figure 14. A version number of the OEF and the specifications of the MDS and the MCS constitute the header part. The node part and link part contain information of the Multimedia Static Schema (MSS). Conceptually, the MSS is equivalent to the MMO hypergraph structure while the MDS and the MCS are derived from the MSS. They reflect the spatial/temporal relations in the MMOs as well as the communication and synchronization requirements.

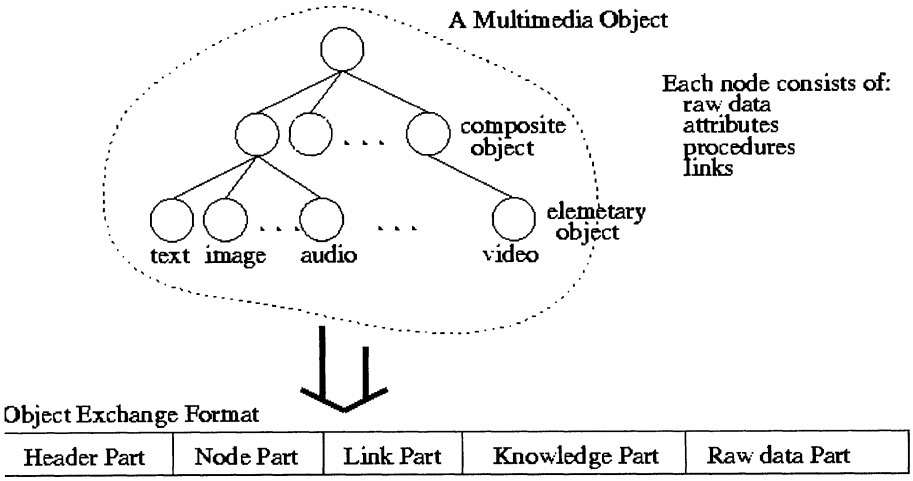

Figure 14. Object Exchange Format.

Our design goal is to formulate a simple yet flexible OEF for our prototype system. It should adapt to the MHEG multimedia object standard [Colai94] easily. To achieve simplicity, the OEF is defined as an ASCII text stream, with blank space as a delimiter. The beginning and ending marks for parts and fields are fixed as three-letter abbreviations. For example, the node part's begin mark is denoted as NPB and the node part's end mark is denoted as NPE.

The simplicity makes it easy to implement the OEF without compromising functionality. The OEF reserves room for future extension by setting NIL fields in order to keep flexibility and extensibility.

6.1.2 Object identification and classification

Five classes are identified when analyzing and designing the OEM. They are mds, mcs, mss, knowledge, and rawdata. Class mss is further decomposed into two classes which are node and link. The top level class mmo contains the above five classes and is the generalization of a composite MMO. An MMO has only one static hypergraph structure, one knowledge structure, and one raw data. Only one MDS and one MCS exist at any point in time. Thus, an mmo class instance consists of only one instance of each of the five classes. The class diagram of the OEM is shown in Figure 15. In the class diagram, there is no *A kind of* relationship among classes. Thus, no inheritance relationships exist in the class hierarchy. There is only the containment relationship in the class hierarchy. For example, class *mmo*

contains class *mms*, *mds*, *mcs*, *knowledge*, and *rawdata*. Class *mss* contains class *node* and *link*.

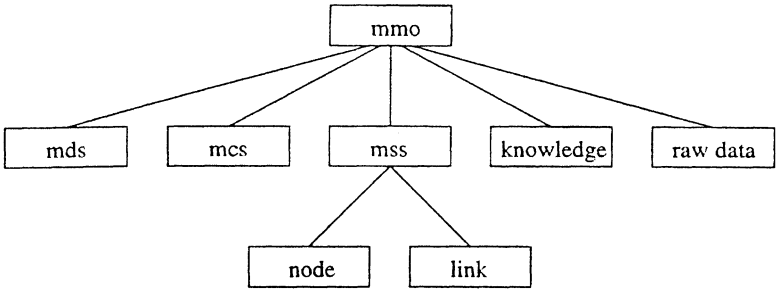

Figure 15. OEM Class Hierarchy.

6.1.3 Class description

The instances of the five classes (*mds*, *mcs*, *mss*, *knowledge*, and *rawdata*) are combined into the MMO represented in Object Exchange Format (OEF). Physically, an MMO in the OEF is an ASCII stream which can be read, written , and transferred. Besides this common data, classes also have their own internal data structure.

6.1.3.1 Class mds and mcs:

The internal data of class mds and mcs are similar. They are G-Net specifications describing the spatial/temporal and synchronization relations within the whole MMO. The basic operations are *put* and *get* - put a G-Net specification into the Object Exchange Format and get a G-Net specification from the Object Exchange Format.

6.1.3.2 Class knowledge:

Like class *mds* and *mcs*, class *knowledge* has only two basic operations: *put* and *get*. The meanings are similar to those in class *mds* and *mcs*. But the internal data are different. Now the internal data is the private knowledge of the MMO, represented as a set of rules, each of which consists of condition part and action part. The rule set can be written in a script language. Therefore, what the *put* or *get* here does is just to put/get the rule set to/from the Object Exchange Format.

6.1.3.3 Class rawdata:

This class manipulates directly on the raw data of objects. It either gets raw data from the OEF, or puts raw data into the OEF. Besides, after getting raw data from the OEF, it can present the data according to its media type. The presentation will be performed by following the specification of the MDS.

6.1.3.4 Class mss:

This class keeps all the information of the MMO hypergraph structure, or MSS. The hypergraph is made of nodes and links, so the class *mss* contains two lower-level classes: *node* and *link* where actual node and link information are maintained. In practice, *mss* maintains two linked lists, one for nodes, one for links. The basic operations are *put* the MSS into the OEF and *get* the MSS from the OEF, *add/delete* a unit to/from the node linked list, and *add/delete* a unit to/from the link linked list. The node and link data are stored in an internal C/C++ structure. The C/C++ structure is instantiated by the MMO editor and shared with the MMO browser in the User Interface module. When the MSS is to be put into the OEF, the C/C++ structure is translated into ASCII stream and vice versa when we get MSS from the OEF.

6.1.3.5 Class mmo:

This is the top-level class that contains the above five classes. It *puts* all the data in its five component classes into the OEF and *gets* data from the OEF and feeds them back into the five classes. Besides, the class *mmo* has a *send* operation which sends the OEF through network by calling MMS network primitive and a *receive* operation which receives the OEF by calling MMR network primitive. It also has *retrieve* operation which fetches an OEF from a remote site and a *present* operation which makes presentation based upon the MDS specification. It also processes the data in the header part of OEF.

6.1.3.6 Class node:

This class contains detailed information of a node in the MMO hypergraph structure. It includes information such as node id, file name, media-type, size, location, etc. Class *rawdata* is defined as *friend* of the class node which means all methods of class *rawdata* can access private data in class node. This is necessary because when accessing the raw data of a node, one has to know such information as media type, size, and location of the raw data. This information is kept in class *node*. So class *rawdata* should have access to private data of class node. It seems that this approach breaches the wall between objects thus violating the principle of information

hiding, but sometimes it is necessary and convenient to do so in the implementation.

6.1.3.7 Class link:

This class contains detailed information of a link in the MMO hypergraph structure. It includes information such as link id, type, starting and ending nodes, direction, etc. Similar to the other classes, the main operations are *put/get* link data to/from the OEF.

6.1.4 User Interface

In the DMS, the OEM does not interact with application users directly. The system user interface provides an interface to edit or browse an MMO hypergraph structure. The OEM is just a call back function of the system user interface. Nevertheless, the OEM is a key module in the DMS which interacts with all other modules. To give advanced users direct control over the OEM, a user interface is developed. Through this user interface, users can create an MMO hypergraph structure, or MSS, by entering node and link information, convert the MSS into OEF, send the OEF over network, receive incoming OEF, etc.

6.2 Interactions between Object Exchange Manager and other modules

In Figure 15, we see that all other modules will interact with the Object Exchange Manager. As the name "Object Exchange" implies, these interactions are for the purpose of exchanging object information. In this section, we will discuss what and how interactions are performed involving the Object Exchange Manager and other modules.

6.2.1 Interaction with User Interface module

Basically, all information about a multimedia object MMO in a specific application domain comes from the application itself. But some information items arrive at the OEM directly from applications, while some are indirectly fed into the OEM. The information of the hypergraph structure is obtained from the User Interface module. In other words, the node and link information in the MMO hypergraph structure are passed to the OEM from the User Interface module. There are two important User Interface submodules: MMO editor and MMO browser. The MMO editor provides an editing environment for creating, modifying, saving, and loading multimedia object hypergraph structure. The MMO browser lets a user navigate through

the MMO hypergraph structure, selecting, viewing, and annotating nodes of interest.

Once a user finishes editing or modifying an MMO, the User Interface module may pass the information of theMMO hypergraph structure (contained in an internal C/C++ structure) to the OEM. Then OEM converts this internal hypergraph structure into an external OEF which is stored in a temporary file in local disk. In practice, the internal C/C++ structure can be shared by the User Interface module and the OEM. The procedures in the OEM could become the callback functions in the User Interface module which is implemented as an X-window application. When an MMO browser in the receiver end is to browse the MMO hypergraph structure, the OEM in the receiver end will have to first extract the hypergraph structure or MSS from the OEF, then convert it to the internal C/C++ structure in the browser. Finally, the browser can browse the hypergraph by accessing the internal C/C++ structure. If a user invokes a browser to view the MMO hypergraph structure created locally, the browser will get the hypergraph directly from the MMO editor instead of retrieving it from the OEM.

6.2.2 Interaction with Integration and Synchronization module

The Integration and Synchronization module derives an MDS from the MSS and generates an MCS as the result of the negotiation with the underlying network layer. Once the MDS and the MCS are available, it calls OEM to store the MDS and MCS data into the OEF. On the other hand, the OEM module can retrieve MDS and MCS data from the OEF and pass them to the Integration and Synchronization module or the Presentation module. So the involved interaction is just passing data back and forth. As we know, MDS and MCS are G-Net specifications in ASCII text form [Chen92]. So the storing and retrieval of MDS and MCS data are trivial.

6.2.3 Interaction with Presentation module

When the OEF arrives at the other end of network, the presentation module needs the information contained in it to present the MMO to users. However, it does not access the OEF directly, it only accesses the internal C/C++ structure converted from the hypergraph structure, or MSS, and MDS embedded in OEF. With MSS and MDS, the presentation module is able to present the MMO in a way intended by the MMO sender.

6.2.4 Interaction with Network Management module

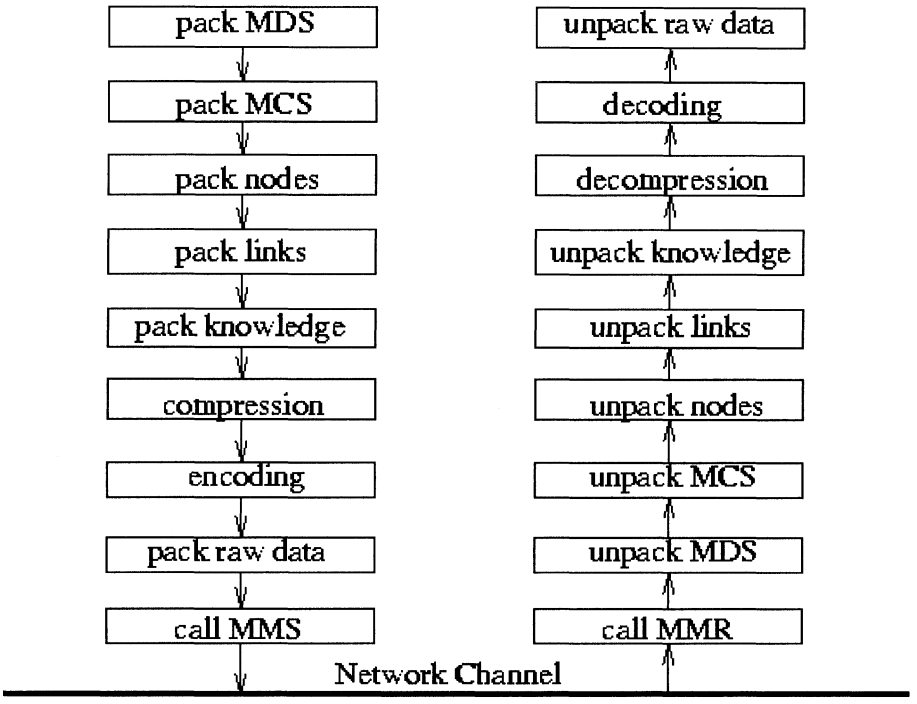

Figure 16. Interaction with Network Management Module.

The interaction with the Network Management module is more complicated. The OEM supports three operations: Send, Receive, and Retrieve which perform the MMO transmission. The Send and Receive operations call primitives MMS (Multimedia Send) and MMR (Multimedia Receive) [Lin94, Lin95] in the Network Management module to perform their functionality. Before sending and receiving, some side jobs are necessary.

The network primitives need the MCS specification from the Integration and Synchronization module. The sending of the MMO is done intelligently in that it is guided by the MCS specification.

An open place and a close place [Lin94, Lin95] are constructed beforehand and the object raw data are embedded between them. After finishing all these side jobs, primitive MMS is called to send the OEF to the receiver's end. On the receiver's end, the receiver is notified of the arrival of any new MMO message. The receiver then performs the Receive operation of the OEM, which in turn, calls primitive MMR in the Network Management module to extract new MMO messages from the receiver's

system mailbox or a specified directory. The Retrieve operation enables a user to retrieve an MMO from a remote site actively, not just waiting passively for any new MMO's arrival. It calls a network primitive *retrieve* to achieve the goal. Basically, the *retrieve* primitive makes use of FTP mechanism to get the remote MMO. It is an operation in the OEM and is written as a C shell script. There is also a similar *retrieve* primitive in the Network Management module.

Before sending a non-text object, such as an image or an audio stream, OEM might compress the raw data using for example JPEG compression algorithm [Walla91]. This greatly reduces the size of the MMO and thus saves the precious network bandwidth and reduces the transmission delay. On receiving such a compressed raw data, the OEM at the receiver's end decompresses the raw data and restores it into original form.

The raw data of a multimedia object might be in binary form. To reduce the overhead of the network layer, the OEM encodes the non-text raw data into ASCII text form. The encoding algorithm used is called *binary encoding* which is used in Columbia University's Imail [Smith92]. This encoding scheme uses only 64 ASCII characters in encoded form. Figure 16 shows the process of constructing, transferring, and retrieving an MMO in OEF.

7. IMPLEMENTATION OF THE EXPERIMENTAL SYSTEM

We have implemented an experimental distributed multimedia system, called the Distributed Active Multimedia System (DAMS), based on the concept of transformation among multimedia schemas and the object exchange manager, as shown in Figure 2. The system is implemented in the X-Window environment on the SUN SPARCstation 5. The object exchange manager is written in C++ programming language, while the other modules are written in C programming language.

On the top level of the system is an application window which provides an integrated user interface for performing various tasks. Illustrated in Figure 17 is a screen dump of the application window. The pull-down menus in the top portion of the window provide functionalities for editing, attributes defining, playback, transformation, packing/unpacking to/from object exchange format, transmission, etc. When composing a multimedia object, the user can choose buttons in the left panel of the window to create links and nodes for constructing the hypergraph structure of the MSS.

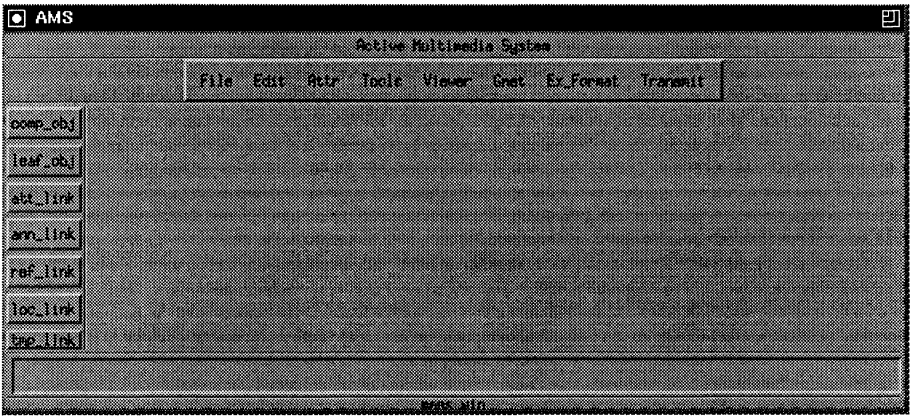

Figure 17. User interface window of the DAMS system.

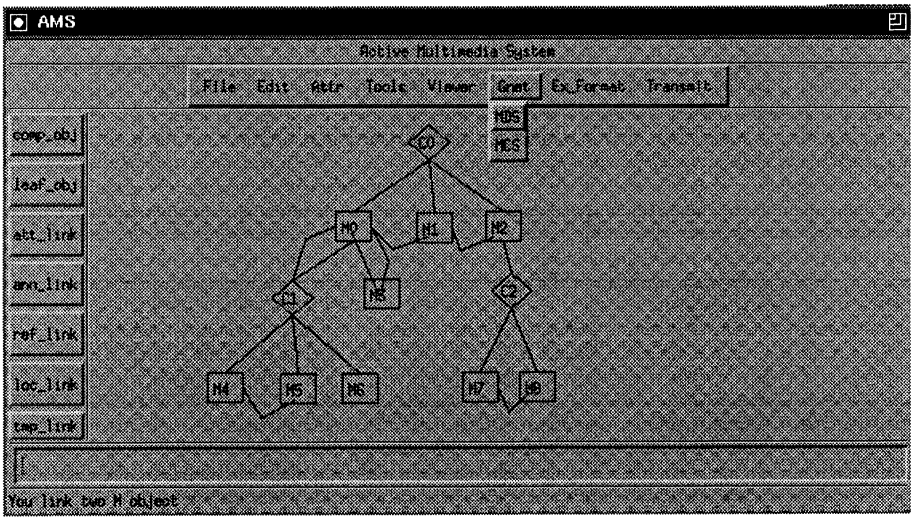

Figure 18. Example of creating a composite object and invoking transformation in the DAMS system.

Referring to the system structure shown in Figure 2, the steps of a typical scenario to compose, view, send, and receive a composite multimedia object in our system are as follows:

1. Editing the hypergraph structure of the MSS: The user can click on *Add* in *Edit* pull-down menu and buttons in the left panel to create nodes and links for the MSS. For example, in Figure 18 a composite object *C*0 composed of basic objects *M*0, *M*1, and *M*2 is created. A *temporal link*

defining the temporal relation between *M*0, *M*1, and *M*2 is created. A *location link* and an *annotation link* defining the spatial relation and the annotation between *M*0 and another composite object *C*1 are also created, respectively. This hypergraph structure corresponds to the one shown in Figure 10a.

2. Defining attribute: In the pop-up dialogue box brought up by clicking on the options in the *Attr* pull-down menu, the user can define the attributes of nodes and links in the MSS, such as media type of an object, the corresponding file name, temporal relation defined by a temporal link, and so on. For example, we can define in Figure 18 the type of *M*0 as image and *M*1 as audio as well as their corresponding file names. For the temporal link linking *M*0, *M*1, and *M*2, the temporal attribute is defined as co-begin.

3. Performing transformation from MSS to MDS: After the MSS is constructed, *MDS* in the *Gnet* pull-down menu, as shown in Figure 18, is chosen to perform the transformation from MSS to MDS.

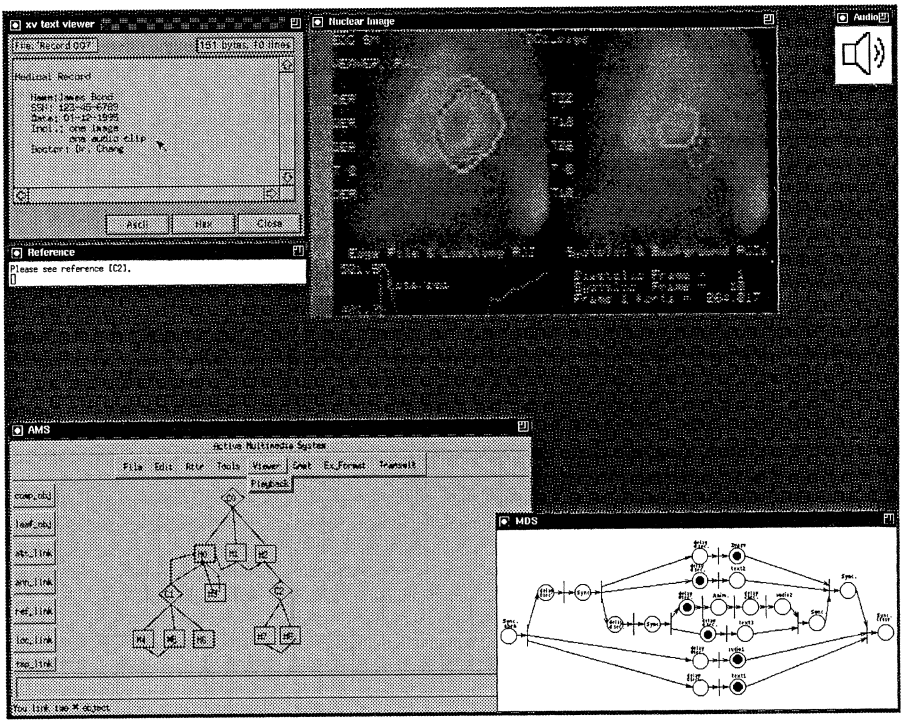

Figure 19. A screen dump of multimedia objects playback in DAMS.

4. Playback: Once the MDS is created, the user can view the presentation of the composite multimedia objects according to the MDS by clicking on *Playback* in *Viewer* pull-down menu. The playback by the

Presentation module is based on the token flow in the MDS. When a token flows to a place in the MDS, the corresponding object is displayed. For example, in Figure 19, a text, an image, and an audio are displayed as there are tokens in their corresponding places. The corresponding MDS is also shown at the lower right corner of the screen. Later on, when tokens flow to $place_{text2}$, $place_{text3}$ and $place_{anim}$, two more texts ("Tumor???" and "OK here") and one animation (the freehand circle) are displayed as the annotation of the image as shown in Figure 20. Note that a reference window with text "Please see reference [C2]" is shown on the screen despite the referenced object *C*2 is not a part of the MDS.

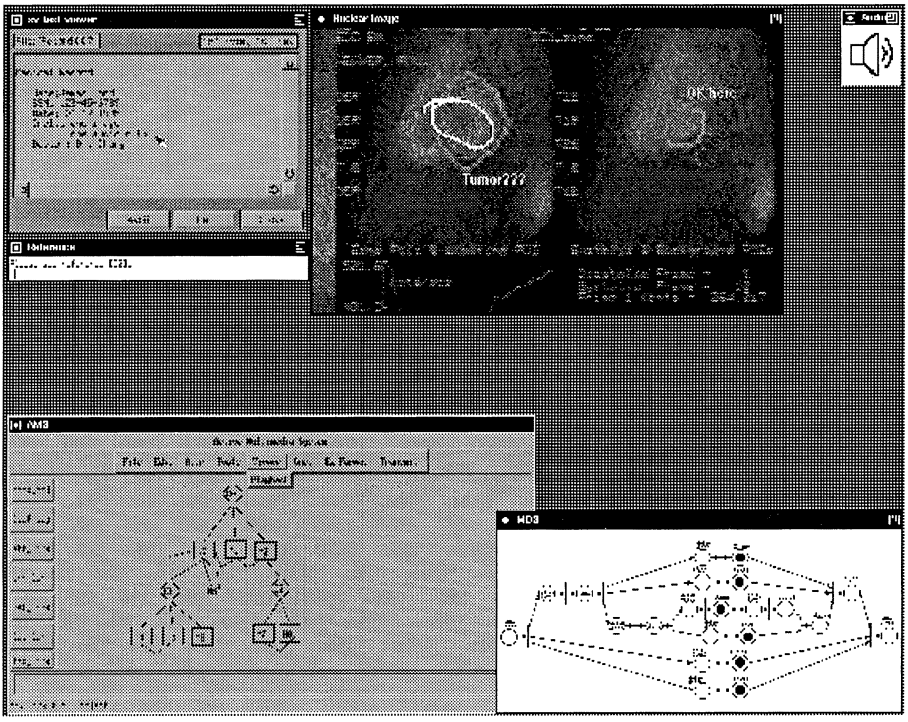

Figure 20. Another screen dump of multimedia objects playback in DAMS.

5. Performing transformation from MDS to MCS: By clicking on *MCS* in the *Gnet* pull-down menu (see Figure 18), the MDS is transformed into MCS.

6. Packing into OEF: The user then packs objects into an OEF file by choosing *Export* in the *Ex_Format* pull-down menu.

7. Sending and/or depositing multimedia objects: The multimedia objects will be sent after the user clicks on *Send* in the *Transmit* pull-down

menu. A Deposit option is also available in this pull-down menu for the user to store the objects in a specified directory.

At the receiving end, the receiver can choose *Receive* in the *Transmit* pull-down menu to retrieve the multimedia object and the corresponding G-Net specification of the MDS from the mailbox. The user can then playback the object according to the MDS.

8. DISCUSSION

In this chapter, we presented models for different schemas in a distributed multimedia system, and algorithms to perform the transformation between them. We then described a unified object exchange format (OEF) for composite multimedia objects.

The hypergraph model provides the static structure of multimedia objects with its rich set of hyperlinks, while the G-Net model provides flexibility and power to combine different aspects of the DMS into an integrated system. The G-Net model also enables a user to specify the MDS and MCS at a high level of abstraction. Furthermore, the G-Net is hierarchical, dynamic, and directly executable [Chen92]. The MSS-to-MDS transformation algorithm transforms the high-level specification of MMOs into a data schema that can be used to perform presentation of MMOs. The MDS-to-MCS transformation algorithm transforms the data schema into the corresponding communication schema that supports the transmission of the objects.

The research issues currently under our investigation include the modeling of interaction, *fuzzy scenarios* [Li94], and other synchronization scenarios; the feasibility of our model on complex applications; detailed specification and evaluation of the QOS parameters; the degradation model and heuristics for the negotiation with the QOS manager; the adaptability of the transformation algorithms in taking the dynamic feedback from the QOS manager to adjust its behavior in order to optimize the usage of the bandwidth in accordance with the reliability of the network; and the heuristics for optimizing the construction of G-Nets that represent the MDS and MCS.

The design and interaction of the Object Exchange Manager in the distributed multimedia system was also described in this chapter. The Object Exchange Manager has the functions of packing all MMO data into OEF and unpacking all data in OEF to construct an active MMO, transferring OEF through network using network primitives provided, and performing relevant

operations during packing/unpacking, such as compression/decompression, and encoding/decoding.

Due to its object oriented design, the OEM has the advantages of easy extension as well as information hiding and reusibilty. Besides, after compression and encoding, an MMO could have a reduced size and could be in ASCII form, making the transmission of the MMO scalible and flexible. The OEM is independent of the underlying network. It can either use the Network Management Module in our system or other network service. Currently, our DAMS is using TCP/IP as the transport service provider, and Ethernet as the underlying network. As this does not meet the needs of real time presentation of an MMO, we plan to investigate and employ a more suitable network service protocol.

Another research issue is the compatibility of our Object Exchange Format to the existing and forthcoming standards, for examples, ISO standards MHEG and PREMO (Presentation Environment for Multimedia Objects) [Herman94]. Our Object Exchange Format will be revised to adapt to the notations of the standards and the Object Exchange Manager will be enhanced to support classes specified in the standards. Since the standards adopts object-oriented approach, we believe that it will be feasible to integrate our OEM into the object class hierarchy of the standards.

APPENDIX A: MSS-TO-MDS ALGORITHM

Procedure MSS-to-MDS (OEF_{MSS}, G_{MDS})
begin
 MSS-to-IM (OEF_{MSS}, *IM*, *L*)
 L = root of *IM*
 newplace(*L*)
 $G_{MDS'}$ = IM-to-MDS' (*R*, *P*)
 MDS'-to-MDS ($G_{MDS'}$, *L*, G_{MDS})
End

Procedure MSS-to-IM (OEF_{MSS}, *IM*, *L*)
 begin
 L = the set of temporal links in OEF_{MSS}
 for each *M* in OEF_{MSS}
 create a node *M'* in *IM*
 N1 = the set of nodes attached to *M*
 N2 = the set of nodes annotating *M*
 while ($N1 \neq \varnothing$)
 remove a node *n* from *N1*

```
    if no node in N1 links to n via a temporal link
      create a node representing n as a child of M'
    else
      create a child Mt for M' in IM
      for each node m in N1 links to n via temporal link S
        create a node representing m as a child of Mt
        N1 = N1 - {m}
      Mt.attribute = S.attribute
  if (N2 ≠ ∅)
    create a node Mt in IM
    M' becomes a child of Mt
    for each node n in N2
      create a node representing n as a child of Mt
end

Function IM-to-MDS'(M, P)
 begin
  if M is a leaf node in IM
    return P
  else
    split(P, P_i, P_j, T_i, T_j)
    /* split place P into two places P_i and P_j and two transitions T_i and T_j,
       where P_i is the input place of T_i and P_j is output place of T_j */
    for each child n of M
      newplace(P_n.delay)
      newplace(P_n.object)
      newtrans(T_n)
      T_tmp = T_i
      add_input_place(T_n, P_n.delay)
      add_output_place(T_n, P_n.object)
      if (M.attribute == PARALLEL)
        add_output_place(T_i, P_n.delay)
        add_input_place(T_j, P_n.object)
      if (M.attribute == SEQUENTIAL)
        add_output_place(T_tmp, P_n.delay)
        if n is the last child processed
          add_input_place(T_j, P_n.object)
        else
          newtrans(T_tmp)
          add_input_place(T_tmp, P_n.object)
      return IM-to-MDS'(n, P_n.object)
end
```

```
Procedure MDS'-to-MDS (G_MDS', L, G_MDS)
 begin
  for each link S=(u, v) in L
    M1 = the place in MDS' representing u
    M2 = the place in MDS' representing v
    newplace(P_aux)
    /* the value of the delay attribute in P_aux is described in Figure 7 */
    P_aux.duration = Table_aux(S.attribute)
    if (S.attribute == SEQUENTIAL)
      newplace(P_aux')
      newtrans(T_aux)
      add_input_place(output_trans(M1), P_aux')
      remove_input_place(output_trans(M1), M1)
      add_output_place(T_aux, P_aux)
      add_output_place(T_aux, P_aux)
      add_input_place(T_aux, M1)
      add_input_place(input_trans(M2), P_aux)
    if (S.attribute == PARALLEL)
      if ((S.sub - attribute == during) or
          ((S.sub - attribute == co-end) and (M1.duration < M2.duration)))
        add_output_place(input_trans(M2), P_aux)
        add_input_place(input_trans(M1), P_aux)
      else
        add_output_place(input_trans(M1), P_aux)
        add_input_place(input_trans(M2), P_aux)
end
```

APPENDIX B: OBJECT EXCHANGE FORMAT

The object exchange format is illustrated in Figure 21.

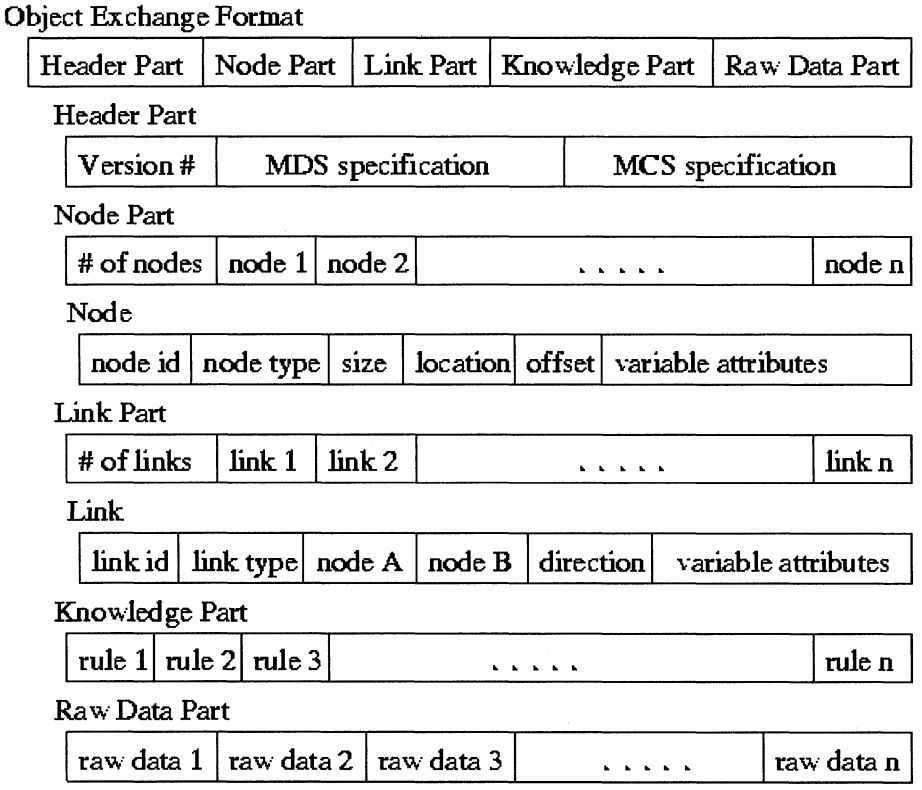

Figure 21. Object Exchange Format.

APPENDIX C: OBJECT EXCHANGE FORMAT: BNF SYNTAX

< Exchange_File > ::= < Begin > < Header_Part > < Node_Part >
< Link_Part > < Knowledge_Part > < Raw_Data_Part > < End >
< Header_Part > ::= < Header_Part_Begin > < Version > < MDS > < MCS >
< Head_Part_End >
< Version > ::= 'DATAEX1.1 '
< MDS > ::= < MDS_Begin > < ascii_spec_of_G-net > < MDS_End >
< MCS > ::= < MCS_Begin > < ascii_spec_of_G-net > < MCS_End >
< Node_Part > ::= < Node_Part_Begin > < Num_of_Nodes > < Node > *
< Node_Part_End >
< Node > ::= < Node_Begin > < Node_Fixed_Attribute >
< Node_Var_Attribute > * < Node_End >
< Node_Fixed_Attribute > ::= < Node_Id > < Node_Type > < Size >
< Location > < Offset > 'NIL '
< Node_Id > ::= < integer > (unique within MMO structure)

< Node_Type > ::= < N_Type > < Media_Type > < Media_Sub_Type >
< Size > ::= < integer > (actual size of node in bytes)
< Location > ::= 'LOCAL ' | < ftp://machine-name/directory-path:file-name >
| 'NIL '
< Offset > ::= < integer > (offset from the beginning of raw data part,
regardless it's LOCAL or not)
< N_Type > ::= < Composite_Type > | < Basic_Media_Type > | 'NIL '
< Media_Type > ::= < Text_Type > | < Image_Type > | < Audio_Type >
| < Video_Type > | ...
< Media_Sub_Type > ::= < Text_Subtype > | < Image_Subtype >
| < Audio_Subtype > | < Video_Subtype > | ...
< Text_Subtype > ::= 'PLAIN ' | 'RICH ' | ...
< Image_Subtype > ::= 'GIF ' | 'JPEG ' | 'PBM ' | 'PGM ' | 'PPM ' | ...
< Audio_Subtype > ::= 'BASIC ' | ...
< Video_Subtype > ::= 'MPEG ' | 'QUICKTIME ' | ...
< Node_Var_Attribute > ::= < Text_Attribute > | < Image_Attribute >
| < Audio_Attribute > | < Video_Attribute > | < Form_Attribute > | 'NIL '
< Text_Attribute > ::= < charset > < Text >
< charset > ::= 'ASCII ' | 'EBCDIC '
< Image_Attribute > ::= 'IMAGE ' < Image_Info >
< Image_Info > ::= 'COMPRESSION_ALG ' | 'PROGRESSIVE_TRANS '
| ...
< Audio_Attribute > ::= 'AUDIO ' < Audio_info >
< Video_Attribute > ::= 'VIDEO ' < Video_info >
< Form_Attribute > ::= < Attribute_Name > < Attribute_Value >
< Link_Part > ::= < Link_Part_Begin > < Num_of_Links > < Link > *
< Link_Part_End >
< Link > ::= < Link_Begin > < Link_Fixed_Attribute >
< Link_Var_Attribute > * < Link_End >
< Link_Fixed_Attribute > ::= < Link_Id > < Link_Type > < Node_List >
< Direction >
< Link_Id > ::= < integer > (unique within MMO structure)
< Link_Type > ::= 'ATTACHMENT ' | 'ANNOTATION ' | 'REFERENCE '
| 'SYNCHRONIZATION ' | 'LOCATION ' | ...
< Node_list > ::= < Node_Id > * (a sequence of integers)
< Direction > ::= < Begin_to_End > | < End_to_Begin > | < Undirected >
< Begin_to_end > ::= 'B '
< End_to_Begin > ::= 'E '
< Undirected > ::= 'U '
< Link_Var_Attribute > ::= < Link_Var_Attr > < Size > < Value >
< Link_Var_Attr > ::= < Attachment_Attribute > | < Annotation_Attribute >
| < Synchronize_Attribute > | < Reference_Attribute > | ...

< Raw_Data_Part > ::= < Raw_Data_Part_Begin > < Raw_Data > *
< Raw_Data_Part_End >
< Raw_Data > ::= < Raw_Data_Begin > < Data > < Raw_Data_End >
< Knowledge_Part > ::= < Knowledge_Part_Begin > < Knowledge > *
< Knowledge_Part_End >
< Knowledge > ::= < Knowledge_Begin > < Rule > * < Knowledge_End >
< Rule > ::= < Rule_Begin > < Condition > < Action > * < Rule_End >
< Begin > ::= 'MMB '
< End > ::= 'MME '
< Header_Part_Begin > ::= 'HPB '
< Header_part_End > ::= 'HPE '
< MDS_Begin > ::= 'MDB '
< MDS_End > ::= 'MDE '
< MCS_Begin > ::= 'MCB '
< MCS_End > ::= 'MCE '
< Node_Part_Begin > ::= 'NPB '
< Node_Part_End > ::= 'NPE '
< Node_Begin > ::= 'NOB '
< Node_End > ::= 'NOE '
< Link_Part_Begin > ::= 'LPB '
< Link_Part_End > ::= 'LPE '
< Link_Begin > ::= 'LIB '
< Link_End > ::= 'LIB '
< Raw_Data_Part_Begin > ::= 'RPB '
< Raw_Data_Part_End > ::= 'RPE '
< Raw_Data_Begin > ::= 'RDB '
< Raw_Data_End > ::= 'RDE '
< Knowledge_Part_Begin > ::= 'KPB '
< Knowledge_Part_End > ::= 'KPE '
< Knowledge_Begin > ::= 'KNB '
< Knowledge_End > ::= 'KNE '
< MDS_End > ::= 'MDE '
< MCS_Begin > ::= 'MCB '
< MCS_End > ::= 'MCE '
< Node_Part_Begin > ::= 'NPB '
< Node_Part_End > ::= 'NPE '
< Node_Begin > ::= 'NOB '
< Node_End > ::= 'NOE '
< Link_Part_Begin > ::= 'LPB '
< Link_Part_End > ::= 'LPE '
< Link_Begin > ::= 'LIB '
< Link_End > ::= 'LIB '

< Raw_Data_Part_Begin > ::= 'RPB '
< Raw_Data_Part_End > ::= 'RPE '
< Raw_Data_Begin > ::= 'RDB '
< Raw_Data_End > ::= 'RDE '
< Knowledge_Part_Begin > ::= 'KPB '
< Knowledge_Part_End > ::= 'KPE '
< Knowledge_Begin > ::= 'KNB '
< Knowledge_End > ::= 'KNE '

Chapter 11

Systems: The Specification of Multimedia Applications

Recent developments in computer technology have enabled large, distributed multimedia applications to be created in such application areas as education [Woolf95], health care [Wong97], and process control [Guha95]. These applications are often web-based and involve a large amount of user interaction. All of these characteristics increase the complexity of designing, coding and testing. The prototyping of multimedia applications based upon software engineering principles has not yet been adequately addressed by the research community although recently research interest in the area of multimedia and software engineering has increased. An indication of this increased interest is the convening of the first International Workshop on Multimedia Software Engineering held in April 1998 as part of the International Conference on Software Engineering [Hirak98]. In this chapter, we apply software engineering methodology to the production of multimedia applications introducing a principled approach to specify, verify, validate and prototype such applications.

Our approach to multimedia application development is based on a collection of tools which support the creation of Teleaction Objects (TAOs) [ChangH95b, Grosk97]. A TAO is a multimedia object with associated hypergraph structure and knowledge structure. The user can create and modify the private knowledge of a TAO so that the TAO will react automatically to certain events. The knowledge structure of a TAO is an active index (IX) [Chang95a] which consists of a collection of index cells (ICs). The hypergraph structure supports the effective presentation and efficient communication of multimedia information. The static aspects of the

hypergraph structure are described by a Multimedia Static Specification (MSS). TAOs are valuable since they greatly improve the selective access and presentation of relevant multimedia information. The tools described in this chapter provide a way to formally specify the TAOs comprising the application, verify and validate the specification, and rapidly prototype the application. The formal specification of the system is based on a Symbol Relation (SR) grammar. Such a multidimensional grammar is particularly attractive since it can describe the spatial and temporal aspects of the application. The specification is converted into TAOML, an extension of HTML.

1. STRUCTURE OF MULTIMEDIA DEVELOPMENT SYSTEM

The structure of the multimedia application development system is shown in Figure 1 below. It mainly consists of two tools. The Formal Specification Tool allows a specification of the MSS to be created. The specification may be either visual or text-based. The specification is then validated using an SR grammar for TAOs. If the specification is valid, the tool generates TAOML and an HTML template for the specified system. The Prototyping Tool includes an IC Builder to create the index cells comprising the knowledge structure of the TAOs. A TAOML interpreter generates HTML code from the TAOML and HTML template and from the information produced by the IC builder. The application generated can then be executed from any web browser working with the distributed IC Manager which controls the active knowledge structure built out of active index cells.

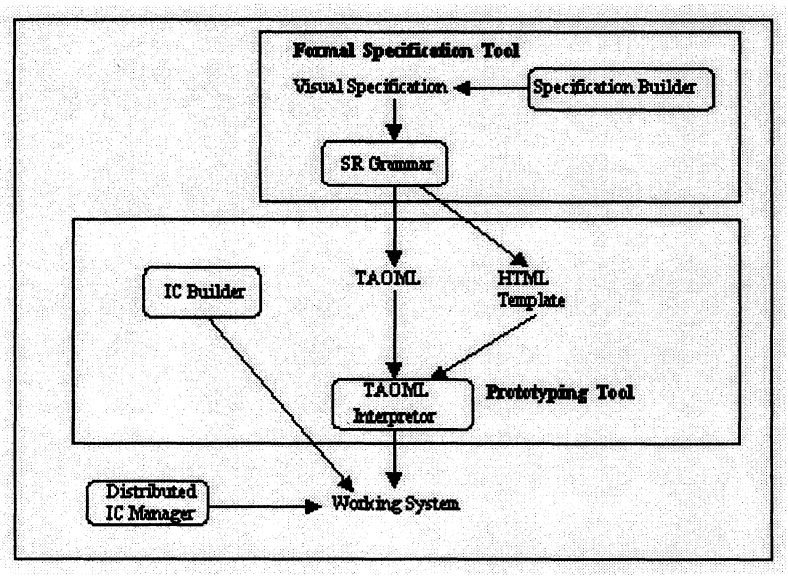

Figure 1. The structure of the multimedia application development environment.

The organization of the rest of this chapter is as follows. The following section reviews the TAO concepts as presented in references [ChangH95b] and [Lin96]. The TAOML extension of HTML is then introduced. This language allows TAO-based systems to be executed from standard web browsers. Section 3 presents a grammatical approach to the formal specification of multimedia applications. Existing multidimensional grammars are analyzed. The Symbol Relation Boundary Grammar (SR Boundary) is chosen as a sufficiently powerful model. The use of this grammar to guide a syntax-directed editor to create the MSS is discussed. A grammar for TAOs is given in section 4. Parsers for this grammar are also discussed. The limitations which must be imposed on the MSS in order to have an efficient (non-exponential) grammar are given. An attributed form of the grammar is introduced in order to associate static knowledge with the MSS. Discussion and future research are given in section 5. The complete grammar for TAOs is given in Appendix B, while the attributed form of the grammar is given in Appendix C.

2. TELEACTION OBJECTS

The use of grammatical formalisms inside of multimedia systems is the most appropriate way to move from a purely manual approach towards an automatic approach [Weitz96a]. The advantages to be gained by this approach include the possibility of introducing a graphical front-end for TAO construction, automatic grammatical checking for the correctness of the structure generated, and introduction of a syntax-directed editor. Finally, the integration of both hypergraph and IX production in a single TAO construction module that produces the hypergraph with the IX attached.

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 in 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 message [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.

Example 1: *Let us consider a kiosk in a train station which presents tourist information about the surrounding area. The opening screen of the presentation played by the kiosk displays an informative message inviting potential users to touch the screen. When a tourist touches the screen a video begins to play along with some background music. Beneath the video, a sequence of text messages describing the video is displayed. At the end of the video, a screen displaying information on local hotels is visualized. After a short time, the initial message is displayed again and the system waits for the next tourist. In figure 2, we show the hypergraph part of the TAO.*

The ICs are attached to the hypergraph to define the actions of the TAO as shown in Figure 3. Index cell IC1 is attached to TAO1, *Welcome*, while index cell IC2 is attached to TAO2, *Display*. IC1 is in the state S0 until the user intervenes with an action by touching the screen. When the message is filtered by the system, it reaches IC1, which passes into state S1 and sends a message to IC2. IC2 passes from the dead state to the active state. It remains in this state until its lifetime is finished or until the user intervenes causing a stop action. This action will cause IC2 to return to the dead state and a message to be sent to IC1 which returns to state S0 where it waits for a new "touch" message.

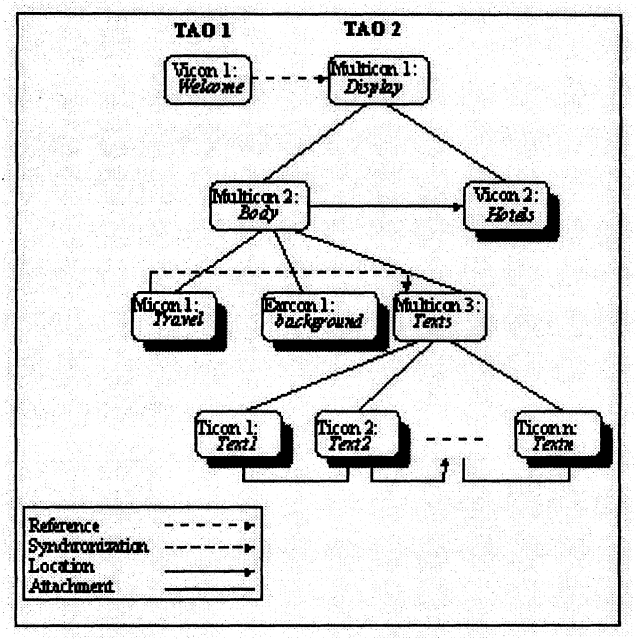

Figure 2. Kiosk hypergraph.

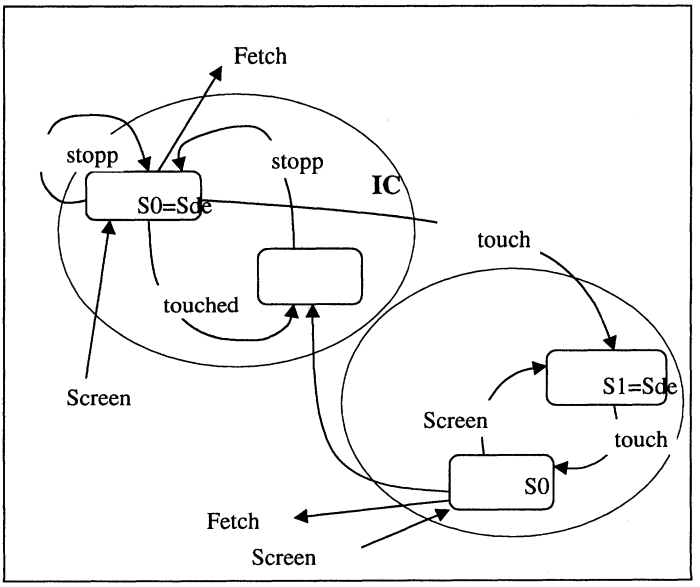

Figure 3. The ICs of the kiosk TAO.

2.1 TAOML

To prototype a distributed multimedia application, 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 on the web server. 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 need only update the interpreter to output another language. The other parts of the 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 structure of a TAO is extended. The new form of the TAO has the following attributes: tao_name, tao_type, p_part, links, ics and sensitivity. These attributes are described below.

- 'tao_name' is the name of the TAO, which is a unique identifier of each TAO.
- 'tao_type' is the media type of TAO - image, text, audio, motion graphs, video or mixed.

- 'p_part' is the physical part of a TAO (see the definition of generalized icon in [Chang87b]). To implement this in the context of TAO_HTML, 'p_part' here can be denoted by an HTML template which indicates the appearance of an HTML page. Templates 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 the TAO.
- 'links' are the links to another TAO. A link has attributes 'link_type' and 'link_obj'. 'link_type' is either relational (spatial or temporal) or structural (COMPOSED OF). In the context of TAO_HTML, a spatial link describes visible relationship between subobjects inside one mixed object. For example, a mixed TAO1 contains an image TAO2 and a text TAO3; then TAO1 has a spatial link with both TAO2 and TAO3. A temporal link usually refers to an invisible object that is not a display element, but whose activation time is influenced by the other TAO. A structural link relates one TAO with another dynamically via user input or external input. For

example, the user clicking a button in TAO1 will invoke another page TAO2; in this case there is a structural link from TAO1 to TAO2.

• 'ic' is 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 at run-time 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 ics. For further details on the meaning of the attributes of the index cells, see [Chang95a].

• 'sensitivity' indicates whether this object is location-sensitive, time-sensitive, content-sensitive or none-sensitive. Then the same object can have different appearances or different functionalities according to the sensitivity. For example, if TAO1 is content-sensitive, it is red when being contained in TAO2 while it is green when being activated by TAO3 via a button. The detailed meaning of sensitivity should be defined by the user according to the requirements of an application. In the newest generation of browsers, sensitivity can be implemented using style sheets.

• 'database' specifies the database that this TAO can access and/or manipulate.

2.2 BNF form for TAO_HTML

The formal definition of the TAO_HTML language is given below.

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 a link relation with other TAOs. It is defined as:
<TAO_REL> "link_name" </TAO_REL>

TAO_HTML Interpreter Algorithm. The TAO_HTML Interpreter translates the TAOML pages into HTML pages so that the user interface is easily implemented using a standard web browser. The TAO_HTML Interpreter is now presented in pseudo-code.

```
procedure interpreter(string TAOname)
begin
open TAO definition file
while (not end of file) do
    begin
        read one line from the file
        recognize tag
get tag information
        store in data structure TAO_struct
    end
call template_parser(TAO_struct)
 end
procedure template_parser(TAO_structure TAO_struct)
begin
  if (IC_PART is specified) then
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) then
  open template file
while (not end of file) do
    begin
        read one line from the file
        if (not <TAO_rel> tag) then
          output html text
        else
        begin
```

```
get link_name from the <TAO_rel> tag
search in the TAO_structure for link_name
if (a link structure is found with
        the same link_name) then
begin
        get link_type and link_TAO_name
        if (link_type=structural) then
              insert <a href..> link in template
              to link with link_TAO_name
          elsif (link_type=spatial) then
          /* insert template of link_TAO_name */
              call interpreter(link_TAO_name)
          end /* if */
    end /* else */
end /* while */
end /* procedure */
```

3. FORMAL SPECIFICATION - THE GRAMMATICAL APPROACH

Formal methods have been proposed as a means for software system designers to assure that a system's requirements accurately reflect the users' requirements and that an implementation is an accurate realization of the design. For these reasons, formal methods provide added reliability to a system. Many researchers claim that formal methods also result in reduced costs since much of the cost of software is a result of imprecision and ambiguity in requirements analysis which necessitates increased testing and maintenance [Saied96]. Formal methods allow a software design to be mathematically modeled and analyzed. A notation for formal specification of a system is provided which can be used to reason about a system in a rigorous manner. In spite of the gains to be realized by adopting formal methods, industrial adoption has been slow. One widely cited impediment to the adaptation of formal methods is the lack of supporting tools.

Due to the complex nature of multimedia applications, they are prime candidates for the application of formal methods. In order to best serve the needs of developers of such applications, we have considered methods specialized for multimedia applications and have settled on a grammatical approach for modeling. Such an approach is well suited to model the hierarchical structure and complex relations of the TAOs composing our

applications. It is also possible to implement tools for the construction, manipulation and analysis of grammatical structures, in this way overcoming one of the most serious impediments to the adaptation of formal methods. These observations are the basis for our selection of a grammatical approach to formal specification of multimedia applications. Formal specification stands as one of the foundations of our approach to the design of multimedia applications, along with the TAO framework and prototyping tools.

Much research has been conducted on multimedia systems and on the interaction between multimedia objects and users [Botto96], however few researchers have used a grammatical approach to specify such systems. One who has is Wittenburg [Weitz94] who proposed a system based on a relational grammar that allows the automatic presentation of multimedia objects. Certain characteristics distinguish his system from ours; in particular, Wittenburg's system does not permit interaction between media objects. The user decides the relations between the media objects that are resolved in a phase of constraint solving. The grammar is used to find the correct values for the physical attributes of the objects in a system in which the user may list the relations to derive without giving the values that the attributes must take. It is not clear, however, how much interaction the user may specify. Finally, due to how it is used, the grammar is directly tied to the parser to be used [Witte92]. This limits the generation of multimedia presentations by the system to those that can be analyzed by the parser.

It is important to make the following observation, which also applies to Wittenburg's relational grammars [Weitz94, Weitz96a, Weitz96b], on the relation between visual grammars and parsers. Today many multidimensional grammars having high generative powers have been produced. This is in contrast to the limited recognizing powers of parsers that are penalized by the high computational complexity of multidimensional grammars. While some researchers see this complexity as a limit on the practical utility of multidimensional grammars, we believe that the parser can be avoided by, for example, using syntax-directed editors. General-purpose editor/browsers offer little assistance to the user, while editor/browsers that identify errors and give users the possibility to redo the errors once they have been identified are more useful. Such editors are syntax-directed and can be used to avoid the complexity of the parser [Costa97b].

3.1 Multidimensional Grammar

After reviewing the existing grammars, we have excluded the more strict context-free grammar models, like Positional Context-Free and Constraint

Multiset Context-Free, because of their limited generative power. In fact, these grammars cannot generate graph languages. Multimedia applications require the handling of complex structures during the parsing phase, therefore a more powerful generative grammar model has to be chosen. However, it is also necessary to avoid increasing the complexity of the parser.

Concerning the complexity of the grammars, a limit on the complexity of parsers of graph grammars has been given by Brandenburg [Brand88] for graph grammars in the *confluence* property. For multidimensional languages, some grammatical derivations that may appear context-free are not since changing the rewrite order of the nonterminals in the derivation can change the final result [Ferru96]. Guaranteeing that the result of all grammatical derivations in a language is independent of the rewrite order of the nonterminals guarantees, by definition, the confluence [Brand88]. This property is indispensable for efficient parsers since any order of application of the rules must result in the recognition of the sentences belonging to the language. If the language is not confluent, any parser must evaluate multiple orderings in order to recognize a sentence.

Then we have turned our attention to multidimensional grammars such as Context-Free Positional Grammars [Costa95b], Constraint Multiset Grammars [Marri96], and Picture Layout Grammars [Golin90]. These grammars use attributes as an essential part of the parsing algorithm since the values of the attributes are crucial for syntactic analysis.

On the other hand, the role of the attributes in the formal structure of a multimedia presentation is primarily to attach semantic knowledge to the grammar model. The knowledge we need to attach may contain information dependent on the application domain as well as information about semantic actions to be triggered. Such knowledge requires a variety of attributes which should also contribute to semantic analysis. This motivation leads us to SR Grammars [Ferru96].

3.2 Symbol Relation Grammars

A common grammatical approach for the description of multidimensional languages uses rewriting mechanisms to generate sentences in the language (e.g. Constraint Multiset Grammars [Marri96], and Picture Layout Grammars [Golin90]). The SR Grammar is one of these grammars. In the SR Grammar formalism [Ferru96], a sentence is viewed as a set of symbol occurrence (s-items) and a set of relation items over symbol occurrences (r-items).

The main feature of SR grammars is that the derivation of a sentence is performed by rewriting both symbol occurrences and relation items by

means of simple context-free style rules. More precisely, during a derivation step a symbol occurrence X^0 in a sentence S^1 is replaced by a sentence S^2, according to a rewriting rule of the form $X^0 \rightarrow S^2$, called an s-item production (s-production). After X^0 has been rewritten, the replacement of the set of r-items involving X^0 is performed through r-item rewriting rules (r-productions) of the form $r(X^0,Y^1) \rightarrow R$, where R is a set of r-items relating Y^1 to s-items in S^2.

In [Ferru96] it has been shown how the notion of attribute context-free grammars may be applied to SR Grammars to implement semantic actions in the language and a boundary version of the SR grammar has been proposed. The Boundary SR Grammar has the confluence property and thus a lower computational complexity. An efficient parser has been given [Ferru96] for confluent languages, which have the connection and limited degree properties, where this last property means that the number of relations that tie one object to another is limited.

4. A BOUNDARY SR GRAMMAR FOR THE TAO HYPERGRAPH STRUCTURE

In this section we describe a Boundary SR Grammar (BSRG) capable of generating the hypergraph structure of the TAO. The grammar is completely general since it does not identify a specific set of relations to be used to construct the TAO. Rather, it permits the instantiation of an arbitrary number of relations since the grammar is defined on base relation types. We identify the following base relation types: spatial; temporal; annotation; reference to the external environment; reference from the external environment. This permits us to use the grammar in various ways, for example, as a module which drives a syntax-directed editor with phases for link creation, assignment of a name to a link, and assignment of a semantic meaning to a link. The grammar is easily specialized for a group of relations for a particular domain. This is possible due to the rewriting of the relations. We exploit this mechanism by having relations represented by non-terminals, which are rewritten with terminal relations only when both terminal nodes involved are reached.

In the subsequent phase of semantic analysis it will be necessary to consider the meaning of the relations, and therefore we introduce an attributed form of the BSRG in which extra information is encapsulated in the attributes attached to the relation.

4.1 A Boundary SR Grammar

The complete version of the BSRG which generates a language that is the set of legal hypergraphs of a TAO is given in Appendix B. The most important rules for the construction of a TAO are briefly described in the following. In order to describe, in a sinthetic way, the productions of the grammar, we will use the symbols z, x to represent, respectively, the elements of the following sets:

$z \in$ {icon, earcon, vicon, ticon, micon}

$x \in$ {icon, vicon, ticon, micon}

• the initial production either directly produces a terminal node or a composite node and a non-terminal node connected by the attachment relation. The only attachment relations derivable are between a multicon (i.e. a composite node) and its children:

1: $S^0 \rightarrow <\{multicon^1, A^1\}\{attach(multicon^1, A^1)\}>$

17: $S^0 \rightarrow <\{x^1\}\ \{\varnothing\}>$ $\quad x \in$ {icon, vicon, ticon, micon}

• it is possible to derive a reference link to and from the external environment i.e.

18: $S^0 \rightarrow <\{x^1, EXT^1\}\{reference(x^1, EXT^1)\}>$

where $z \in$ {icon, earcon, vicon, ticon, micon} and EXT represents the external environment. Productions 2-16, 18-24, 58-67 describe the external reference to the TAO.

• The annotation relations have as a parent node any base or composite node, but must have as child node the composite node of a new TAO annotating the preceding node, i.e.:

43: $A^0 \rightarrow <\{x^1, A^1, S^1\}\{rel\ (x^1, A^1)\ annotation(x^1, S^1)\}>$

See productions 43-47, 51-57 and 64-67.

• The spatial and temporal relations are derived via the *rel* relation which is rewritten when a terminal is involved. Productions 25-26, 29, 35-36, 39, 43-44, 47, 53-54, 56-57 produce the *rel* relation; i.e:

29: $A^0 \rightarrow <\{x^1, A^1\}\{rel\ (x^1, A^1)\ rel\ (A^1, x^1)\}>$

which can be rewritten by using the following rewriting rules:

R64: $rel\ (x^0, A^0) \rightarrow$ [25,26,27,28,29,43,44,45,46,47] $\{y(x^0, x^1)\}$

R65: $rel\ (A^0, x^0) \rightarrow$ [25,26,27,28,29,43,44,45,46,47] $\{y(x^1, x^0)\}$

where $x \in$ {icon, vicon, ticon, micon, multicon} and $y \in$ {synchronization, location}

• For the earcon, rewritings with spatial relations are forbidden (see productions 30-34, 48-52); i.e.:

34: $A^0 \rightarrow <\{earcon^1, A^1\}\{synchronization\ (earcon^1, A^1)\ synchronization\ (A^1, earcon^1)\}>$

R66: $rel\ (x^0, A^0) \rightarrow$ [30,31,32,33,34,48,49,50,51,52] $\{synchronization(x^0, earcon^1)\}$

R67: rel $(A^0, x^0) \rightarrow$ [30,31,32,33,34,48,49,50,51,52] {synchronization(earcon^1, x^0)}

• the spatial, temporal and reference relations are derived at a high level of derivation. These may be duplicated, rewritten and located in any point of a TAO except when we wish to derive a bundled node. In the case of a bundled node we use productions 40-41:

40: $A^0 \rightarrow <\{B^1\}\ \{\varnothing\}>$

41: $B^0 \rightarrow <\{A^1\}\ \{\varnothing\}>$

There are no rewriting rules for the relations after the application of the above productions. As a consequence, the sentential forms of the language which we obain do not have reference links, location links, or synchronization links which involve nodes both external to and internal to the bundle.

The sequence of derivations steps used to generate the TAO of example 1, is shown in Appendix A. In the example the nonterminal symbols of each derivation step, which need to be rewritten later, are shown in bold. The relations which involve these symbols and therefore have not yet been rewritten, are also in bold. At each step of the derivation, next to the $\Rightarrow$ symbol, we indicate the s-production and the r-production(s) which have been involved in the derivation step.

4.2 Parsing

As stated in Section 3.1, the confluence property is important for multidimensional languages since if the language satisfies this property an efficient parser for the language can be produced. A Boundary SR grammar must satisfy two constraints in order to be confluent - the graphs generated by the language must be connected and each node must have a limited degree (i.e. the number of links from each node must be less than or equal to some constant).

The hypergraphs generated by the Boundary SR grammar for TAOs given in Appendix B are connected since the hypergraph is uniquely given by the derivation tree with root S, the start symbol. If we have two TAOs, these TAOs may be connected by a reference link. This reference link is the point of connectivity between the two TAOs.

The limited connectivity property is also satisfied, even if we are not able to give a priori a limit. It is reasonable to assume (since it doesn't limit the type of TAOs we wish to generate) that a node is linked only to its neighbors. Further, there is a constant number of link types. Therefore, even if a node is connected to its neighbors via all link types there is not a linear

degree of connectivity. This limitation drastically reduces the size of the language, but it does not disallow the sentences (i.e. TAOs) that we are interested in.

5. SEMANTIC EXTENSION OF TAO USING ATTRIBUTE SR GRAMMMARS

Teleaction objects consist of a hypergraph representing the interface of a multimedia application with an associated knowledge structure. We have shown how the hypergraph structure can be generated by an SR grammar. The knowledge structure is an active index and is created by using the IC builder tool. It is possible to extend the SR grammar for TAOs to include the associated knowledge as semantic actions associated with the grammar. An extension of the SR grammar which does this was proposed in [Ferru94]. The extended SR grammar is an Attribute SR Grammar which associates a set of inherited and/or synthesized attributes with the non-terminal symbols of V_N and with the symbols of the relations of V_R, associating evaluation rules with the s- and r-productions. This permits the use of different knowledge in different contexts, using the context-sensitive generative power of the derivations to pass the attributes.

6. ENVIRONMENTAL ADAPTABILITY

Since the grammatical model provides a variety of relations, the knowledge will provide the correct routines for materialization/interpretation of these relations. We can give the same names to different routines that work with diverse media types and, by attribute passing, let the correct routines reach the terminals. This approach allows compatibility with diverse multimedia environments in which a relation may have many meanings.

Knowledge is expressed as references to an area of memory of the distributed ICs. The correct knowledge base will be loaded in this area of memory. Since we separate the construction phase of the environment from the construction phase of a TAO, multiple TAO systems may be constructed in the same environment. Furthermore, diverse environments may be supported by loading an environment-specific knowledge base. For this, it is sufficient to use only inherited attributes [Aho86]. Given the limited nature of their use, a dependence among the attributes of parents and children only is assured, thus assuring the acyclic nature of the dependence graph and the

efficiency of the calculations (a top down visit using the hypergaph attachments is sufficient, or from the generative point of view, a top down visit of the derivation tree).

The attribute scheme for a TAO is given in [Arndt97b].

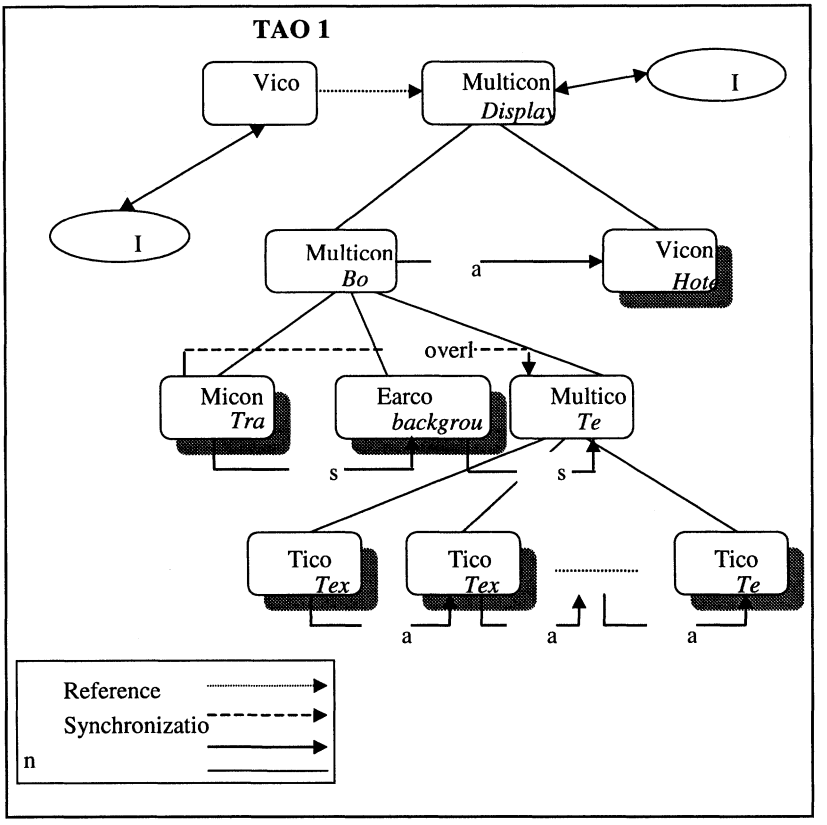

Figure 4. Hypergraph structure with attached knowledge.

7. DISCUSSION

We have presented the basis for a principled approach to the production of distributed multimedia applications. The unifying principle for our approach is the Teleaction Object. TAOML, an extension of HTML has been introduced to allow distributed multimedia applications to be

prototyped using standard web browsers as a front-end and the distributed IC manager to manage the knowledge associated with the application. A boundary SR grammar has been introduced to allow for the formal specification of TAOs. The interpreter for TAOML as well as the distributed IC manager and a graphical front-end for specifying TAOs have been developed. In the future, we will develop a syntax-directed editor based on the grammar. This editor will produce TAOML output via semantic actions. This output will then be fed to the interpreter, providing a unified approach to application development. We also plan to investigate the use of the formal specification to prove properties of the application.

APPENDIX A: DERIVATION STEPS OF KIOSK TAO

TAO 1 :

S^0

$\Rightarrow$18 <{vicon^1, **EXT^1**} {**reference**(vicon^1, **EXT^1**)}>

$\Rightarrow$58,R156 <{vicon^1, ext^1} {reference(vicon^1, ext^1)}>

TAO 2:

S^0

$\Rightarrow$3 <{multicon^1, **A^1**, **EXT^1**} {**attach**(multicon^1, **A^1**) **reference**(**EXT^1**, multicon^1)}>

$\Rightarrow$58,R156 <{multicon^1, **A^1**, ext^1} {**attach**(multicon^1, **A^1**)} {reference(ext^1, multicon^1)}>

$\Rightarrow$ 35,R1 <{multicon^1, {multicon^2, **A^2**, **A^3**}, ext^1} {**rel**(multicon^2, **A^2**) **attach**(multicon^2, **A^3**) **attach**(multicon^1, **A^2**) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

$\Rightarrow$28,R64,R25 <{multicon^1, {multicon^2, {vicon^2}, **A^3**}, ext^1} {rel(multicon^2, vicon^2) **attach**(multicon^2, **A^3**) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

$\Rightarrow$40,R23 <{multicon^1, {multicon^2, {vicon^2}, {**B^1**}}, ext^1} {rel(multicon^2, vicon^2) **attach**(multicon^2, **B^1**) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

$\Rightarrow$41,R24 <{multicon^1, {multicon^2, {vicon^2}, {**A^4**}}, ext^1} {rel(multicon^2, vicon^2) **attach**(multicon^2, **A^4**) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

$\Rightarrow$36,R1 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, **A^5**, **A^6**}}, ext^1} {**rel**(**A^5**, multicon^3) **attach**(multicon^3, **A^6**) rel(multicon^2, vicon^2) **attach**(multicon^2, **A^5**) attach(multicon^2, multicon^3) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

$\Rightarrow$42,R61,R19 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, {**A^7**}, **A^6**}}, ext^1}

{**rel**($\mathbf{A^7}$, multicon^3) **rel**($\mathbf{A^7}$, multicon^3) **attach**(multicon^3, $\mathbf{A^6}$) rel(multicon^2, vicon^2) **attach**(multicon^2, $\mathbf{A^7}$) attach(multicon^2, multicon^3) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

⇒25,R65,R63,R28 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, {micon^1, $\mathbf{A^8}$}, $\mathbf{A^6}$}}, ext^1} {**rel**(micon^1, $\mathbf{A^8}$) location(micon^1, multicon^3) **rel**($\mathbf{A^8}$, multicon^3) **attach**(multicon^3, $\mathbf{A^6}$) rel(multicon^2, vicon^2) attach(multicon^2, micon^1) **attach**(multicon^2, $\mathbf{A^8}$) attach(multicon^2, multicon^3) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

⇒33,R66,R67,R25 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, {micon^1, {earcon^1}}, $\mathbf{A^6}$}}, ext^1} {synchronization(micon^1, earcon^1) location(micon^1, multicon^3) synchronization(earcon^1, multicon^3) **attach**(multicon^3, $\mathbf{A^6}$) rel(multicon^2, vicon^2) attach(multicon^2, micon^1) attach(multicon^2, earcon^1) attach(multicon^2, multicon^3) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

⇒25,R28 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, {micon^1, {earcon^1}}, {ticon^1, $\mathbf{A^9}$}}, ext^1} {**rel**(ticon^1, $\mathbf{A^9}$) synchronization(micon^1, earcon^1) location(micon^1, multicon^3) synchronization(earcon^1, multicon^3) attach(multicon^3, ticon^1) **attach**(multicon^3, $\mathbf{A^9}$) rel(multicon^2, vicon^2) attach(multicon^2, micon^1) attach(multicon^2, earcon^1) attach(multicon^2, multicon^3) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

⇒25,R64,R28 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, {micon^1, {earcon^1}}, {ticon^1, {ticon^2, $\mathbf{A^{10}}$}}}}, ext^1} {**rel**(ticon^2, $\mathbf{A^{10}}$) synchronization(ticon^1, ticon^2) synchronization(micon^1, earcon^1) location(micon^1, multicon^3) synchronization(earcon^1, multicon^3) attach(multicon^3, ticon^1) attach(multicon^3, ticon^2) **attach**(multicon^3, $\mathbf{A^{10}}$) rel(multicon^2, vicon^2) attach(multicon^2, micon^1) attach(multicon^2, earcon^1) attach(multicon^2, multicon^3) attach(multicon^1, vicon^2) attach(multicon^1, multicon^2) reference(ext^1, multicon^1)}>

.....

⇒25,R64,R28 <{multicon^1, {multicon^2, {vicon^2}, {multicon^3, {micon^1, {earcon^1}}, {ticon^1, {ticon^2, {.....{ticon^{n-1}, $\mathbf{A^{n-1+8}}$}....}, ext^1} {**rel**(ticon^{n-1}, $\mathbf{A^{n-1+8}}$) synchroniza-

tion(ticon^{n-2}, ticon^{n-1}) ..... synchronization(ticon^{2}, ticon^{3}) synchronization(ticon^{1}, ticon^{2}) synchronization(micon^{1}, earcon^{1}) location(micon^{1}, multicon^{3}) synchronization(earcon^{1}, multicon^{3}) attach(multicon^{3}, ticon^{1}) attach(multicon^{3}, ticon^{2}) ..... attach(multicon^{3}, ticon^{n-2}) **attach**(multicon^{3}, **A**$^{n-1+8}$) rel(multicon^{2}, vicon^{2}) attach(multicon^{2}, micon^{1}) attach(multicon^{2}, earcon^{1}) attach(multicon^{2}, multicon^{3}) attach(multicon^{1}, vicon^{2}) attach(multicon^{1}, multicon^{2}) reference(ext^{1}, multicon^{1})}>

⇒28,R64,R26 <{multicon^{1}, {multicon^{2}, {vicon^{2}}, {multicon^{3}, {micon^{1}, {earcon^{1}}}, {ticon^{1}, {ticon^{2}, {.....{ticon^{n-1}, ticon^{n}}....}, ext^{1}} {rel(ticon^{n-1}, ticon^{n}) synchronization(ticon^{n-2}, ticon^{n-1}) ..... synchronization(ticon^{2}, ticon^{3}) synchronization(ticon^{1}, ticon^{2}) synchronization(micon^{1}, earcon^{1}) location(micon^{1}, multicon^{3}) synchronization(earcon^{1}, multicon^{3}) attach(multicon^{3}, ticon^{1}) attach(multicon^{3}, ticon^{2}) ..... attach(multicon^{3}, ticon^{n-2}) attach(multicon^{3}, ticon^{n-1}) attach(multicon^{3}, ticon^{n}) rel(multicon^{2}, vicon^{2}) attach(multicon^{2}, micon^{1}) attach(multicon^{2}, earcon^{1}) attach(multicon^{2}, multicon^{3}) attach(multicon^{1}, vicon^{2}) attach(multicon^{1}, multicon^{2}) reference(ext^{1}, multicon^{1})}>

APPENDIX B: THE BOUNDARY SYMBOL RELATION GRAMMAR FOR THE TAO

The BSRG G for the TAO is defined as follow. G= (V_N, V_T, V_R, S, P, R) where S is the start symbol, the set of nonterminals is V_N = {S, A, B, EXT}, the set of terminals is V_T = {icon, vicon, earcon, ticon, micon, multicon, ext} and the set of relations is V_R = {rel, attach, annotation, synchronization, location, reference}. The terminal *ext* represent the icons that have an external reference to or from them.

Notation: *in order to avoid duplication of productions which differ by just one terminal symbol, we introduce the symbols x, z, t , y to represent, respectively, the symbols of the following sets:*

x∈{icon, vicon, ticon, micon}

z∈{icon, earcon, vicon, ticon, micon}

t ∈{icon, earcon, vicon, ticon, micon, multicon}

y∈{synchronization, location}

P:

1: $S^0 \rightarrow <\{multicon^1, A^1\}\{attach(multicon^1, A^1)\}>$

2: $S^0 \rightarrow <\{multicon^1, A^1, EXT^1\}\{attach(multicon^1, A^1)$ $reference(multicon^1, EXT^1)\}>$

3: $S^0 \rightarrow <\{multicon^1, A^1, EXT^1\}\{attach(multicon^1, A^1)$ $reference(EXT^1, multicon^1)\}>$

4: $S^0 \rightarrow <\{multicon^1, A^1, EXT^1\}\{attach(multicon^1, A^1)$ $reference(multicon^1, EXT^1)$
$reference(EXT^1, multicon^1)\}>$

5: $S^0 \rightarrow <\{multicon^1, A^1, ext^1\}\{attach(multicon^1, A^1)$ $reference(A^1, ext^1)\}>$

6: $S^0 \rightarrow <\{multicon^1, A^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $reference(multicon^1, EXT^1)$ $reference(A^1, ext^1)\}>$

7: $S^0 \rightarrow <\{multicon^1, A^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $reference(EXT^1, multicon^1)$ $reference(A^1, ext^1)\}>$

8: $S^0 \rightarrow <\{multicon^1, A^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $reference(multicon^1, EXT^1)$
$reference(EXT^1, multicon^1)$ $reference(A^1, ext^1)\}>$

9: $S^0 \rightarrow <\{multicon^1, A^1, S^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)\}>$

10: $S^0 \rightarrow <\{multicon^1, A^1, S^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$
$reference(multicon^1, EXT^1)\}>$

11: $S^0 \rightarrow <\{multicon^1, A^1, S^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$
$reference(EXT^1, multicon^1)\}>$

12: $S^0 \rightarrow <\{multicon^1, A^1, S^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$
$reference(multicon^1, EXT^1)$ $reference(EXT^1, multicon^1)\}>$

13: $S^0 \rightarrow <\{multicon^1, A^1, S^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$ $reference(A^1, ext^1)\}>$

14: $S^0 \rightarrow <\{multicon^1, A^1, S^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$
$reference(multicon^1, EXT^1)$ $reference(A^1, ext^1)\}>$

15: $S^0 \rightarrow <\{multicon^1, A^1, S^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$
$reference(EXT^1, multicon^1)$ $reference(A^1, ext^1)\}>$

16: $S^0 \rightarrow <\{multicon^1, A^1, S^1, EXT^1, ext^1\}\{attach(multicon^1, A^1)$ $annotation(multicon^1, S^1)$
$reference(multicon^1, EXT^1)$ $reference(EXT^1, multicon^1)$ $reference(A^1, ext^1)\}>$

17: $S^0 \rightarrow <\{x^1\}\ \{\varnothing\}>$ $\quad x \in \{icon, vicon, ticon, micon\}$

18: $S^0 \rightarrow <\{x^1, EXT^1\}\{reference(x^1, EXT^1)\}>$

19: $S^0 \rightarrow <\{EXT^1, x^1\}\{reference(EXT^1, x^1)\}>$
20: $S^0 \rightarrow <\{x^1, EXT^1\}\{reference(EXT^1, x^1)\ reference(EXT^1, x^1)\}>$
21: $S^0 \rightarrow <\{x^1, S^1\}\{annotation(x^1, S^1)\}>$
22: $S^0 \rightarrow <\{x^1, S^1, EXT^1\}\{annotation(x^1, S^1)\ reference(x^1, EXT^1)\}>$
23: $S^0 \rightarrow <\{x^1, S^1, EXT^1\}\{annotation(x^1, S^1)\ reference(EXT^1, x^1)\}>$
24: $S^0 \rightarrow <\{x^1, S^1, EXT^1\}\{annotation(x^1, S^1)\ reference(x^1, EXT^1)\ reference(EXT^1, x^1)\}>$
25: $A^0 \rightarrow <\{x^1, A^1\}\{rel\ (x^1, A^1)\}>$
26: $A^0 \rightarrow <\{x^1, A^1\}\{rel\ (A^1, x^1)\}>$
27: $A^0 \rightarrow <\{x^1, A^1\}\{\varnothing\}>$
28: $A^0 \rightarrow <\{x^1\}\ \{\varnothing\}>$
29: $A^0 \rightarrow <\{x^1, A^1\}\{rel\ (x^1, A^1)\ rel\ (A^1, x^1)\}>$
30: $A^0 \rightarrow <\{earcon^1, A^1\}\{synchronization\ (earcon^1, A^1)\}>$
31: $A^0 \rightarrow <\{earcon^1, A^1\}\{synchronization\ (A^1, earcon^1)\}>$
32: $A^0 \rightarrow <\{earcon^1, A^1\}\{\varnothing\}>$
33: $A^0 \rightarrow <\{earcon^1\}\ \{\varnothing\}>$
34: $A^0 \rightarrow <\{earcon^1, A^1\}\{synchronization\ (earcon^1, A^1)\ synchronization\ (A^1, earcon^1)\}>$
35: $A^0 \rightarrow <\{multicon^1, A^1, A^2\}\{rel(multicon^1, A^1)\ attach(multicon^1, A^2)\}>$
36: $A^0 \rightarrow <\{multicon^1, A^1, A^2\}\{rel(A^1, multicon^1)\ attach(multicon^1, A^2)\}>$
37: $A^0 \rightarrow <\{multicon^1, A^1, A^2\}\{attach(multicon^1, A^2)\}>$
38: $A^0 \rightarrow <\{multicon^1, A^2\}\{attach(multicon^1, A^2)\}>$
39: $A^0 \rightarrow <\{multicon^1, A^1, A^2\}\{rel(multicon^1, A^1)\ rel(A^1, multicon^1)\ attach(multicon^1, A^2)\}>$

40: $A^0 \rightarrow <\{B^1\}\ \{\varnothing\}>$
41: $B^0 \rightarrow <\{A^1\}\ \{\varnothing\}>$ /*The productions 40 and 41 generate the bundled nodes. In fact, there are not existing r-productions for the relation *rel*, which include these productions. As a consequence, the sentential forms of the language which we obain do not have reference links, location links, or synchronization links which involve nodes both external to and internal to the bundle. We have r-productions only for the attachment links.*/
42: $A^0 \rightarrow <\{A^1\}\ \{\varnothing\}>$

43: $A^0 \rightarrow <\{x^1, A^1, S^1\}\{rel\ (x^1, A^1)\ annotation(x^1, S^1)\}>$
44: $A^0 \rightarrow <\{x^1, A^1, S^1\}\{rel\ (A^1, x^1)\ annotation(x^1, S^1)\}>$
45: $A^0 \rightarrow <\{x^1, A^1, S^1\}\{annotation(x^1, S^1)\}>$
46: $A^0 \rightarrow <\{x^1, S^1\}\{annotation(x^1, S^1)\}>$
47: $A^0 \rightarrow <\{x^1, A^1, S^1\}\{rel\ (x^1, A^1)\ rel\ (A^1, x^1)\ annotation(x^1, S^1)\}>$

48: $A^0 \rightarrow$ <{earcon^1, A^1, S^1}{synchronization (earcon^1, A^1) annotation(earcon^1, S^1)}>
49: $A^0 \rightarrow$ <{earcon^1, A^1, S^1}{synchronization (A^1, earcon^1) annotation(earcon^1, S^1)}>
50: $A^0 \rightarrow$ <{earcon^1, A^1, S^1}{annotation(earcon^1, S^1)}>
51: $A^0 \rightarrow$ <{earcon^1, S^1}{annotation(earcon^1, S^1)}>
52: $A^0 \rightarrow$ <{earcon^1, A^1, S^1}{synchronization (earcon^1, A^1) synchronization (A^1, earcon^1)
annotation(earcon^1, S^1)}>

53: $A^0 \rightarrow$ <{multicon^1, A^1, A^2, S^1}{rel(multicon^1, A^1) attach(multicon^1, A^2) annotation(multicon^1, S^1)}>
54: $A^0 \rightarrow$ <{multicon^1, A^1, A^2, S^1}{rel(A^1, multicon^1) attach(multicon^1, A^2) annotation(multicon^1, S^1)}>
55: $A^0 \rightarrow$ <{multicon^1, A^1, A^2, S^1}{attach(multicon^1, A^2) annotation(multicon^1, S^1)}>
56: $A^0 \rightarrow$ <{multicon^1, A^2, S^1}{rel(multicon^1, A^2) annotation(multicon^1, S^1)}>
57: $A^0 \rightarrow$ <{multicon^1, A^1, A^2, S^1}{rel(multicon^1, A^1) rel(A^1, multicon^1) attach(multicon^1, A^2)
annotation(multicon^1, S^1)}>

58: $EXT^0 \rightarrow$ <{ext^1} {∅}>
59: $EXT^0 \rightarrow$ <{EXT^1} {∅}>

60: $S^0 \rightarrow$ <{earcon^1} {∅}>
61: $S^0 \rightarrow$ <{earcon^1, EXT^1}{reference(earcon^1, EXT^1)}>
62: $S^0 \rightarrow$ <{earcon^1, EXT^1}{reference(EXT^1, earcon^1)}>
63: $S^0 \rightarrow$ <{earcon^1, EXT^1}{reference(earcon^1, EXT^1) reference(EXT^1, earcon^1)}>
64: $S^0 \rightarrow$ <{earcon^1, S^1}{annotation(earcon^1, S^1)}>
65: $S^0 \rightarrow$ <{earcon^1, S^1, EXT^1}{annotation(earcon^1, S^1) reference(earcon^1, EXT^1)}>
66: $S^0 \rightarrow$ <{earcon^1, S^1, EXT^1}{annotation(earcon^1, S^1) reference(EXT^1, earcon^1)}>
67: $S^0 \rightarrow$ <{earcon^1, S^1, EXT^1}{annotation(earcon^1, S^1) reference(earcon^1, EXT^1) reference(EXT^1, earcon^1)}>

R:
R1: attach(multicon^0, A^0) → [35,36,37,39,53,54,55,57] {attach(multicon^0, A^1) attach(multicon^0, multicon^1)}
R2: attach(multicon^0, A^0) → [35,36,37,39,53,54,55,57] {attach(multicon^0,

A^1) attach($multicon^0$, $multicon^1$)
y($multicon^1$, $multicon^0$)}
R3: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^0$, $multicon^1$)}
R4: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^1$, $multicon^0$) y($multicon^0$, $multicon^1$)}

R5: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
rel($multicon^0$, A^1)}
R6: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
rel(A^1, $multicon^0$)}
R7: attach($multicon^0$, A^0) → [35,36,37,39,5354,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$) rel($multicon^0$, A^1) rel(A^1, $multicon^0$)}
R8: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$) y($multicon^0$, $multicon^1$) rel($multicon^0$, A^1)}

R9: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^0$, $multicon^1$) rel(A^1, $multicon^0$)}
R10: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$) y($multicon^0$, $multicon^1$) rel($multicon^0$, A^1) rel(A^1, $multicon^0$)}
R11: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^1$, $multicon^0$) rel($multicon^0$, A^1)}
R12: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^1$, $multicon^0$) rel(A^1, $multicon^0$)}
R13: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^1$, $multicon^0$) rel($multicon^0$, A^1) rel(A^1, $multicon^0$)}
R14: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$)
y($multicon^1$, $multicon^0$) y($multicon^0$, $multicon^1$) rel($multicon^0$, A^1)}
R15: attach($multicon^0$, A^0) → [35,36,37,39,53,54,55,57] {attach($multicon^0$, A^1) attach($multicon^0$, $multicon^1$) y($multicon^1$, $multicon^0$) y($multicon^0$, $multicon^1$) rel(A^1, $multicon^0$)}

R16: attach(multicon^0, A^0) → [35,36,37,39,53,54,55,57] {attach(multicon^0, A^1) attach(multicon^0, multicon^1) y(multicon^1, multicon^0) y(multicon^0, multicon^1) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R17: attach(multicon^0, A^0) → [38,56] {attach(multicon^0, multicon^1)}

R18: attach(multicon^0, A^0) → [38,56] {attach(multicon^0, multicon^1) y(multicon^0, multicon^1)}

R19: attach(multicon^0, A^0) → [42] {attach(multicon^0, A^1)}

R20: attach(multicon^0, A^0) → [42] {attach(multicon^0, A^1) rel(multicon^0, A^1)}

R21: attach(multicon^0, A^0) → [42] {attach(multicon^0, A^1) rel(A^1, multicon^0)}

R22: attach(multicon^0, A^0) → [42] {attach(multicon^0, A^1) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R23: attach(multicon^0, A^0) → [40] {attach(multicon^0, B^1)}

R24: attach(multicon^0, B^0) → [41] {attach(multicon^0, A^1)}

R25: attach(multicon^0, A^0) → [28,33,46,51] {attach(multicon^0, z^1)}

R26: attach(multicon^0, A^0) → [28,46] {attach(multicon^0, x^1)} y(multicon^0, x^1)}

R27: attach(multicon^0, A^0) → [33,51] {attach(multicon^0, earcon^1)} synchronization(multicon^0, earcon^1)}

R28: attach(multicon^0, A^0) → [25,26,27,29,30,31,32,34,43,44,45,47,48,48,50,52] {attach(multicon^0, z^1) attach(multicon^0, A^1)}

R29: attach(multicon^0, A^0) → [25,26,27,29,30,31,32,34,43,44,45,47,48,48,50,52] {attach(multicon^0, z^1) attach(multicon^0, A^1) rel(multicon^0, A^1)}

R30: attach(multicon^0, A^0) → [25,26,27,29,30,31,32,34,43,44,45,47,48,48,50,52] {attach(multicon^0, z^1) attach(multicon^0, A^1) rel(A^1, multicon^0)}

R31: attach(multicon^0, A^0) → [25,26,27,29,30,31,32,34,43,44,45,47,48,48,50,52] {attach(multicon^0, z^1) attach(multicon^0, A^1) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R32: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1) y(multicon^0, x^1)}

R33: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1) y(multicon^0, x^1) rel(multicon^0, A^1)}

R34: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1) y(multicon^0, x^1) rel(A^1, multicon^0)}

R35: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(multicon^0, x^1) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R36: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0)}

R37: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) rel(multicon^0, A^1)}

R38: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) rel(A^1, multicon^0)}

R39: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R40: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) y(multicon^0, x^1)}

R41: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) y(multicon^0, x^1) rel(multicon^0, A^1)}

R42: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) y(multicon^0, x^1) rel(A^1, multicon^0)}

R43: attach(multicon^0, A^0) → [25,26,27,29,43,44,45,47] {attach(multicon^0, x^1) attach(multicon^0, A^1)
y(x^1, multicon^0) y(multicon^0, x^1) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R44: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1)}

R45: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1)
rel(multicon^0, A^1)}

R46: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1)
rel(A^1, multicon^0)}

R47: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1)
rel(multicon^0, A^1) rel(A^1, multicon^0)}

R48: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1)}

R49: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) rel(multicon^0, A^1)}

R50: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) rel(A^1, multicon^0)}

R51: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R52: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(earcon^1, multicon^0)}

R53: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(earcon^1, multicon^0) rel(multicon^0, A^1)}

R54: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(earcon^1, multicon^0) rel(A^1, multicon^0)}

R55: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(earcon^1, multicon^0) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R56: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) synchronization(earcon^1, multicon^0)}

R57: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) synchronization(earcon^1, multicon^0) rel(multicon^0, A^1)}

R58: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) synchronization(earcon^1, multicon^0) rel(A^1, multicon^0)}

R59: attach(multicon^0, A^0) → [30,31,32,34,48,49,50,52] {attach(multicon^0, earcon^1) attach(multicon^0, A^1) synchronization(multicon^0, earcon^1) synchronization(earcon^1, multicon^0) rel(multicon^0, A^1) rel(A^1, multicon^0)}

R60: rel (x^0, A^0) → [42] {rel (x^0, A^1) rel (x^0, A^1)}
R61: rel (A^0, x^0) → [42] {rel (A^1, x^0) rel (A^1, x^0)}

R62: rel (x^0, A^0) → [25,26,27,29,30,31,32,34,35,36,37,39,42,43,44,45,47,48,49,50,52,53,54,55, 57]
{rel (x^0, A^1)}
R63: rel (A^0, x^0) → [25,26,27,29,30,31,32,34,35,36,37,39,42,43,44,45,47,48,49,50,52,53,54,55, 57]
{rel (A^1, x^0)}

R64: rel (x^0, A^0) → [25,26,27,28,29,43,44,45,46,47] {y(x^0, x^1)}
R65: rel (A^0, x^0) → [25,26,27,28,29,43,44,45,46,47] {y(x^1, x^0)}

R66: rel (x^0, A^0) → [30,31,32,33,34,48,49,50,51,52] {synchronization(x^0, $earcon^1$)}
R67: rel (A^0, x^0) → [30,31,32,33,34,48,49,50,51,52] {synchronization($earcon^1$, x^0)}

R68: rel (x^0, A^0) → [35,36,37,38,39,53,54,55,56,57] {rel (x^0, A^2)}
R69: rel (A^0, x^0) → [35,36,37,38,39,53,54,55,56,57] {rel (A^2, x^0)}

R70: rel (x^0, A^0) → [35,36,37,38,39,53,54,55,56,57] {y (x^0, $multicon^1$)}
R71: rel (A^0, x^0) → [35,36,37,38,39,53,54,55,56,57] {y ($multicon^1$, x^0)}

R72: rel (x^0, A^0) → [43,44,45,46,47,53,54,55,56,57] {rel (x^0, S^1)}
R73: rel (A^0, x^0) → [43,44,45,46,47,53,54,55,56,57] {rel (S^1, x^0)}

R74: synchronization ($earcon^0$, A^0) → [25,26,27,28,29,30,31,32,33,34,43,44,45,46,47,48,49,50,51,52]
{synchronization($earcon^0$, z^1)}
R75: synchronization (A^0, $earcon^0$) → [25,26,27,28,29,30,31,32,33,34,43,44,45,46,47,48,49,50,51,52]
{synchronization(z^1, $earcon^0$)}
R76: synchronization ($earcon^0$, A^0) → [25,26,27,29,30,31,32,34,35,36,37,39,42,43,44,45,47,48,49,50,52,53, 54,55,56,57] {synchronization($earcon^0$, A^1)}
R77: synchronization (A^0, $earcon^0$) → [25,26,27,29,30,31,32,34,35,36,37,39,42,43,44,45, 47,48,49,50,52,53, 54,55,56,57] {synchronization(A^1, $earcon^0$)}

R78: synchronization (earcon^0, A^0) → [35,36,37,38,39,53,54,55,56,57] {synchronization(earcon^0, A^2)}
R79: synchronization (A^0, earcon^0) → [35,36,37,38,39,53,54,55,56,57] {synchronization(A^2, earcon^0)}

R80: synchronization (earcon^0, A^0) → [48,49,50,51,52,53,54,55,56,57] {synchronization(earcon^0, S^1)}
R81: synchronization (A^0, earcon^0) → [48,49,50,51,52,53,54,55,56,57] {synchronization(S^1, earcon^0)}

R82: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]{annotation(multicon^0, multicon^1)}
R83: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1)}
R84: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
y(multicon^1, multicon^0)}
R85: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1) y(multicon^1, multicon^0)}

R86: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
rel(multicon^0, A^1)}
R87: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1) rel(multicon^0, A^1)}
R88: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
y(multicon^1, multicon^0) rel(multicon^0, A^1)}
R89: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1) y(multicon^1, multicon^0) rel(multicon^0, A^1)}

R90: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)
rel(A^1, multicon^0)}
R91: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {annotation(multicon^0, multicon^1)

y(multicon^0, multicon^1) rel(A^1, multicon^0)}
R92: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(multicon^0, multicon^1)
y(multicon^1, multicon^0) rel(A^1, multicon^0)}
R93: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1) y(multicon^1, multicon^0) rel(A^1, multicon^0)}
R94: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(multicon^0, multicon^1)
rel(A^1, multicon^0) rel(multicon^0, A^1)}
R95: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1) rel(A^1, multicon^0) rel(multicon^0, A^1)}
R96: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(multicon^0, multicon^1)
y(multicon^1, multicon^0) rel(A^1, multicon^0) rel(multicon^0, A^1)}
R97: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(multicon^0, multicon^1)
y(multicon^0, multicon^1) y(multicon^1, multicon^0) rel(A^1, multicon^0)
rel(multicon^0, A^1)}

R98: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)}
R99: annotation(multicon^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)
y(z^0, multicon^1)}
R100: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)
y(multicon^1, z^0)}
R101: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)
y(z^0, multicon^1) y(multicon^1, z^0)}
R102: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1) rel(z^0, A^1)}
R103: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)
y(z^0, multicon^1) rel(z^0, A^1)}
R104: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)
y(multicon^1, z^0) rel(z^0, A^1)}
R105: annotation(z^0, S^1) → [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
{annotation(z^0, multicon^1)

$y(z^0, multicon^1)$ $y(multicon^1, z^0)$ $rel(z^0, A^1)$}

R106: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $rel(A^1, z^0)$}
R107: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $y(z^0, multicon^1)$ $rel(A^1, z^0)$}
R108: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $y(multicon^1, z^0)$ $rel(A^1, z^0)$}
R109: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $y(z^0, multicon^1)$ $y(multicon^1, z^0)$ $rel(A^1, z^0)$}

R110: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $rel(A^1, z^0)$ $rel(z^0, A^1)$}
R111: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $y(z^0, multicon^1)$ $rel(A^1, z^0)$ $rel(z^0, A^1)$}
R112: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $y(multicon^1, z^0)$ $rel(A^1, z^0)$ $rel(z^0, A^1)$}
R113: $annotation(z^0, S^1) \rightarrow$ [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] {$annotation(z^0, multicon^1)$ $y(z^0, multicon^1)$ $y(multicon^1, z^0)$ $rel(A^1, z^0)$ $rel(z^0, A^1)$}
R114: $annotation(multicon^0, S^1) \rightarrow$ [17,18,19,20,21,22,23,24] {$annotation(multicon^0, x^1)$}
R115: $annotation(multicon^0, S^1) \rightarrow$ [17,18,19,20,21,22,23,24] {$annotation(multicon^0, x^1)$ $y(multicon^0, x^1)$}
R116: $annotation(multicon^0, S^1) \rightarrow$ [17,18,19,20,21,22,23,24] {$annotation(multicon^0, x^1)$ $y(x^1, multicon^0)$}
R117: $annotation(multicon^0, S^1) \rightarrow$ [17,18,19,20,21,22,23,24] {$annotation(multicon^0, x^1)$ $y(x^1, multicon^0)$ $y(multicon^0, x^1)$}
R118: $annotation(multicon^0, S^1) \rightarrow$ [60,61,62,63,64,65,66,67] {$annotation(multicon^0, x^1)$}
R119: $annotation(multicon^0, S^1) \rightarrow$ [60,61,62,63,64,65,66,67] {$annotation(multicon^0, x^1)$ $synchronization(multicon^0, earcon^1)$}
R120: $annotation(multicon^0, S^1) \rightarrow$ [60,61,62,63,64,65,66,67] {$annotation(multicon^0, x^1)$

synchronization(earcon^1, multicon^0)}

R121: annotation(multicon^0, S^1) → [60,61,62,63,64,65,66,67] {annotation(multicon^0, x^1) synchronization(earcon^1, multicon^0) synchronization(multicon^0, earcon^1)}

R122: annotation(z^0, S^1) → [17,18,19,20,21,22,23,24] {annotation(z^0, x^1)}

R123: annotation(z^0, S^1) → [17,18,19,20,21,22,23,24] {annotation(z^0, x^1) y(x^1, z^0)}

R124: annotation(z^0, S^1) → [17,18,19,20,21,22,23,24] {annotation(z^0, x^1) y(z^0, x^1)}

R125: annotation(z^0, S^1) → [17,18,19,20,21,22,23,24] {annotation(z^0, x^1) y(x^1, z^0) y(z^0, x^1)}

R126: annotation(z^0, S^1) → [60,61,62,63,64,65,66,67] {annotation(z^0, earcon^1)}

R127: annotation(z^0, S^1) → [60,61,62,63,64,65,66,67] {annotation(z^0, earcon^1) synchronization(earcon^1, z^0)}

R128: annotation(z^0, S^1) → [60,61,62,63,64,65,66,67] {annotation(z^0, earcon^1) synchronization(z^0, earcon^1)}

R129: annotation(z^0, S^1) → [60,61,62,63,64,65,66,67] {annotation(z^0, earcon^1) synchronization(earcon^1, z^0) synchronization(z^0, earcon^1)}

R130: reference(ext, A^0) → [40] {reference(B^1, ext)}

R131: reference(B^0, ext) → [41] {reference(A^1, ext)}

R132: reference(ext, A^0) → [25,26,27,29,30,31,32,34,35,36,37,39,42,43,44,45,47,48,49,50,52,53,54,55,57] {reference(ext, A^1)}

R133: reference(ext, A^0) → [42] {reference(ext, A^1) reference(ext, A^1)}

R134: reference(ext, A^0) → [25,26,27,28,29,30,31,32,33,34,43,44,45,46,47,48,49,50,51,52] {reference(z^1, ext)}

R135: reference(ext, A^0) → [25,26,27,28,29,30,31,32,33,34,43,44,45,46,47,48,49,50,51,52] {reference(ext, z^1)}

R136: reference(ext, A^0) → [25,26,27,29,30,31,32,34,43,44,45,47,48,49,50,52] {reference(ext, A^1) reference(z^1, ext)}

R137: reference(ext, A^0) → [25,26,27,29,30,31,32,34,43,44,45,47,48,49,50,52] {reference(ext, A^1)

reference(ext, z^1)}

R138: reference(ext, A^0) → [35,36,37,38,39,53,54,55,56,57] {reference(multicon^1, ext)}

R139: reference(ext, A^0) → [35,36,37,38,39,53,54,55,56,57] {reference(ext, multicon^1)}

R140: reference(ext, A^0) → [35,36,37,39,53,54,55,57] {reference(ext, A^1) reference(multicon^1, ext)}

R141: reference(ext, A^0) → [35,36,37,39,53,54,55,57] {reference(ext, A^1) reference(ext, multicon^1)}

R142: reference(ext, A^0) → [35,36,37,38,39,53,54,55,56,57] {reference(ext, A^2) reference(multicon^1, ext)}

R143: reference(ext, A^0) → [35,36,37,38,39,53,54,55,56,57] {reference(ext, A^2) reference(ext, multicon^1)}

R144: reference(ext, A^0) → [35,36,37,38,39,53,54,55,56,57] {reference(ext, A^2)}

R145: reference(ext, A^0) → [35,36,37,39,53,54,55,56,57] {reference(ext, A^1) reference(ext, A^2)}

R146: reference(A^0, ext) → [25,26,27,28,29,30,31,32,33,34] {reference(z^1, ext)}

R147: reference(A^0, ext) → [42] {reference(A^1, ext) reference(A^1, ext)}

R148: reference(A^0, ext) → [25,26,27,29,30,31,32,34,35,36,37,39,42,43,44,45,47,48,49,50,52,53,54,55,57]
{reference(A^1, ext)}

R149: reference(A^0, ext) → [25,26,27,29,30,31,32,34,43,44,45,47,48,49,50,52] {reference(A^1, ext) reference(z^1, ext)}

R150: reference(A^0, ext) → [35,36,37,38,39,53,54,55,56,57] {reference(multicon^1, ext)}

R151: reference(A^0, ext) → [35,36,37,38,39,53,54,55,56,57] {reference(A^2, ext)}

R152: reference(A^0, ext) → [35,36,37,38,39,53,54,55,56,57] {reference(multicon^1, ext), reference(A^2, ext)}

R153: reference(A^0, ext) → [35,36,37,39,53,54,55,57] {reference(multicon^1, ext) reference(A^1, ext)}

R154: reference(A^0, ext) → [35,36,37,39,53,54,55,57] {reference(multicon^1, ext) reference(A^1, ext) reference(A^2, ext)}

R155: reference(A^0, ext) → [35,36,37,39,53,54,55,57] {reference(A^1, ext) reference(A^2, ext)}

R156: reference(t^0, EXT^0) → [58] {reference(t^0, ext^1)} t ∈ {multicon, z}

R157: reference(t^0, EXT^0) → [59] {reference(t^0, EXT^1)}

R158: reference(t^0, EXT^0) → [59] {reference(t^0, EXT^1) reference(t^0, EXT^1)}

R159: reference(EXT^0, t^0) → [58] {reference(ext^1, t^0)}

R160: reference(EXT^0, t^0) → [59] {reference(EXT^1, t^0)}

R161: reference(EXT^0, t^0) → [59] {reference(EXT^1, t^0) reference(EXT^1, t^0)}

APPENDIX C: THE ATTRIBUTED BOUNDARY SYMBOL RELATION GRAMMAR FOR THE TAO

Given the grammar of Appendix B we will add an inherited attribute x.K to each x terminal in V_T and nonterminal in V_N to represent the knowledge associated with each node of the hypergraph; an attribute x.name to each nonterminal in V_N and synchronization and location relation in V_R. The inherited attribute represents a reference to the file that will contain the nonterminal or the relation descriptions respectively. We also add the following sets of semantic rules to each of the productions in P.

For production 1 to 8:

$multicon^1$.K := *load(system knowledge)*

A^1.K := *load(system knowledge)* ∪ S^0.K (this semantic rule is not valid for production 1)

$multicon^1$.name := *load(name)*

For production 17 to 20:

z^1.K := *load(system knowledge)* ∪ *load(private knowledge)*

$z^1.name^0$:= *load(name)*

For production 25 to 34:

z^1.K := *load(private knowledge)* ∪ A^0.K

A^1.K := A^0.K (this semantic rule is not valid for production 28 and 33)

For production 35 to 39:

$multicon^1$.K := *load(environment knowledge)* ∪ A^0.K

A^1.K := A^0.K (this semantic rule is not valid for production 39)

A^2.K := *load(environment knowledge)* ∪ A^0.K

$multicon^1.name^0$:= *load(name)*

For production 40:

B^1.K := A^0.K

For production 41:

$A^1.K := B^0.K$

For production 42:
$A^1.K := A^0.K$

For production 9 to 16:
$multicon^1.K :=$ *load(system knowledge)*
$A^1.K :=$ *load(system knowledge)*
$multicon^1.name^0 :=$ *load(name)*

For production 21 to 24:
$z^1.K :=$ *load(system knowledge)* $\cup S^0.K \cup$ *load(private knowledge)*
$z^1.name^0 :=$ *load(name)*

For production 44 to 52:
$z^1.K :=$ *load(private knowledge)* $\cup A^0.K$
$A^1.K := A^0.K$
$z^1.K :=$ *load(name)*
For production 53 to 57:
$multicon^1.K :=$ *load(environment knowledge)* $\cup A^0.K$
$A^1.K := A^0.K$
$A^2.K :=$ *load(environment knowledge)* $\cup A^0.K$
$multicon^1.name^0 :=$ *load(name)*

We will add the attribute **synchronization(a^i,a^j).name** and **location(a^i,a^j).name** to the *synchronization* and *location* relations to specify the instance that hold between a^i and a^j.

We will add the attribute **reference(ext,a^j).name** and **reference(a^j,ext).name** to the *reference* relation to specify the name of the external icon that will be involved in the relation together with a^j.

We will add the attribute **annotation(ext,a^j).name** to the *annotation* relation to specify the name of the icon that will represent the root of the TAO annotated to a^j.

We also add the following sets of semantic rules to each production in R that contains the respective r-item:

$y(multicon^0, multicon^1).name :=$ *load(name)*
$y(multicon^1, multicon^0).name :=$ *load(name)*
$y(multicon^0, x^1).name :=$ *load(name)*
$y(x^1, multicon^0).name :=$ *load(name)*
$synchronization(multicon^0, earcon^1).name :=$ *load(name)*
$synchronization(earcon^1, multicon^0).name :=$ *load(name)*
$y(x^0, x^1).name :=$ *load(name)*

y(x^1, x^0).name:= *load(name)*
synchronization(x^0, $earcon^1$).name:= *load(name)*
synchronization($earcon^1$, x^0).name:= *load(name)*
synchronization($earcon^0$, z^1).name:= *load(name)*
synchronization(z^1, $earcon^0$).name:= *load(name)*
reference(z^1, ext).name:= *load(name)*
reference(ext, z^1).name:= *load(name)*
reference($multicon^1$, ext).name:= *load(name)*
reference(ext, $multicon^1$).name:= *load(name)*
annotation($multicon^0$, $multicon^1$).name:= $multicon^1$.name
annotation ($multicon^0$, z^1).name:= z^1.name
annotation (z^0, $multicon^1$).name:= $multicon^1$.name
annotation (z^0, z^1).name:= z^1.name

Chapter 12

Exercises and Project Suggestions

EXERCISE 1:

USA Today on December 31, 1998 carried an interesting article, "Birth of a New Order", talking about the year that world's lines of time and space collapsed. The year is of course 1998. The most incisive paragraphs are excerpted below:

The global, time-crunched market driven by electronic information "forces things to get bigger and smaller at the same time," says Nicholas Negroponte, author and technologist at the Massachusetts Institute of Technology. "And that's so ironic, when things want to do both but not stay in the middle. There will be an increasing absence of things that aren't either very local or very global". Oil and cars aren't much suited to being small and local. So they're moving to become gigantic and cross-border.

As for being small and local, that's where the Internet, or World Wide Web, comes in -- and it works in two ways. It lets little companies be global, so a start-up in a garage can put its goods or services on a Web site and sell worldwide, competing against midsize or big companies, wiping out disadvantages (such as distribution and scope) that once had to do with distance. And since little companies can change direction faster than bigger ones, they have an advantage in time. Big companies used to have time and distance on their side. Increasingly, little ones do.

And so in 1998, we had the phenomenon of Amazon.com, which has become such a symbol of small beating big that business people have turned it into a verb: to be "amazoned".

In the context of the above, write a mini-essay to discuss what do you envision multimedia software engineering will become, and how multimedia software engineering might help the "little guys" compete against the "big guys", or the other way around. The mini-essay should be between 1,500 and 2,000 words, with no less than 3 and no more than 5 references. A "template" is provided:

1st paragraph: My vision of multimedia software engineering in the year 200X.
2nd paragraph: My vision of a small company in the year 200X.
3rd paragraph: A scenario of the small company in action.
4th paragraph: Reasons why MSE can help the small company compete against the big guys.
5th paragraph: More discussions.
3 to 5 references.

The mini-essay should be e-mailed to the instructor by the deadline.

EXERCISE 2:

The purpose of this exercise is to enable the students to gain familiarity with the active index approach to active information system design. As discussed in the book, the hypermedia model and the active index together can be used to model active distributed multimedia information systems. In this exercise we will first concentrate on the active index component.

Let us consider an adaptive distance learning system.

The distance learning materials are organized into a hypermedia structure. A student user can browse through these multimedia documents and follow the links to access related multimedia documents. As such, the hypermedia structure is passive, waiting to be accessed by the user.

We can make the hypermedia structure active by associating index cells with selected multimedia documents. The idea is to designate a special document so that when many students access this document, it means they have reached a certain level of proficiency and therefore the learning materials should be adjusted to become more difficult. Likewise when many students access a special document indicating deficiency, it means they have problems and therefore the learning materials should be made simpler.

The following index cell types are specified:

Proficiency-level index cell: The proficiency-level index cell is associated with a certain specific multimedia document (such as doc-1, usually reacheable only by proficient students). When this index cell is

triggered, it will increase the proficiency-level by 1. When the proficiency-level has reached a predefined threshold (such as 3), it will send message to the instructor, informing the instructor that a sufficient number of students have reached this level of proficiency. It will also send messages to certain documents (such as doc-3, doc-4, doc-5) to become harder.

Deficiency-level index cell: The deficiency-level index cell is associated with a certain specific multimedia document (such as doc-2, usually reacheable only by deficient students). When this index cell is triggered, it will increase the deficiency-level by 1. When the deficiency-level has reached a predefined threshold (such as 2), it will send message to the instructor, informing the instructor that a significant number of students have reached this level of deficiency. It will also send messages to certain documents (such as doc-3, doc-4, doc-5) to become easier.

Self-adjustment index cell: This self-adjustment index cell is associated with multimedia documents containing learning materials (such as doc-3, doc-4 and doc-5). When it receives a "harder" message, it upgrades the learning materials to become harder. Likewise, when it receives a "easier" message, it downgrades the learning materials to become easier.

The above are three index cell types. The instances can be associated with individual multimedia documents (such as doc-1, ..., doc-5).

There is also a home page (such as doc-0), with links to the other documents (such as doc-1, ..., doc-5).

(a) Draw state-transition diagrams to define graphically the three index cell types.

(b) Specify the three index cell types formally using mathematical notations ic = (X, Y, S, s_o, A, t_{max}, f, g).

(c) Draw a diagram showing a few multimedia documents (such as doc-1, ..., doc-5) enhanced with the index cells to illustrate how these index cells work together to form an active index system.

(d) Use the IC Builder to construct the three index cell types. The output from IC Builder, together with the appropriate actions (C functions) and specification of input message space, output message space, will become input to the IC Compiler to generate the IC Manager.

How to download the IC Builder:

Please go to the author's web site at: www.cs.pitt.edu/~chang and follow the links to multimedia software engineering courseware. In a directory for IC_Builder the following files can be found: ictapp.zip and ictype.zip. Use pkunzip to unzip and install under Windows.

At the time of the writing of this book, the above mentioned files are at:
http://www.cs.pitt.edu/~jung/IC_Builder/ictapp.zip
http://www.cs.pitt.edu/~jung/IC_Builder/ictype.zip

Additional Explanation:

For this exercise there is NO NEED to write any C functions. The assignment can be handed in either as a hard copy or via the Internet. For the part where you use IC_Builder to construct IC types, you can turn in the output file(s) generated by the IC_Builder, which are ascii files X.in. You can also provide screen dumps captured during the construction process. If you use Internet, it will be the best if you can provide the URL so that the instructor can browse the web pages containing the solutions. In other words, please prepare a set of web pages and figures can be embedded as gif/jpg files. This will be the easiest for other people to read. It will also be useful when you later develop a presentation based upon such materials.

EXERCISE 3:

The purpose of this exercise is to understand the relationship between active index and Petri nets. Both are tools for the modeling of distributed multimedia systems. Active index cells are added incrementally to build a dynamic index, and the connections can also change dynamically. However, if the massages passed between index cells are deterministically routed, then it is possible to convert active index into a Petri net. Otherwise you must use a Petri net with conditions (predicates) associated with the transitions, or an Evaluation Net (E-net).

(a) Convert the active index you constructed in Exercise #2 into a Petri net (or an E-net).

(b) Take the diagram you drew in part (c) of Exercise #2. Redraw it here (because you may want to make some changes), and now use the marked Petri net to illustrate the scenario. You can draw a sequence of marked Petri net to show how the system works.

Additional Explanation:

(a) If we consider how the active index system passes messages and reaches equilibrium state (if one exists), this leads to a formal study using, for instance, the Petri net model.

(b) Notice this is the beginning of a systematical approach to build prototypes for active distributed multimedia systems. Can we create a new systematic approach, i.e., a new software process model, for distributed multimedia systems design?

(c) The index cells could span several nodes. Therefore, the active index system is a distributed index. The IC Managers must also be distributed to the nodes in the networks.

EXERCISE 4:

The purpose of this exercise is to experiment with MICE, the prototyping tool for distributed multimedia computing. The MICE development environment provides step-by-step instructions on how to use the IC_Builder to create the Ics, how to use the IC_Compiler to create the customized IC_Manager, and finally how to generate the multimedia application.

First, you need to compile the index cell specifications using the IC_Compiler explained in Section 3 of Chapter 8. The IC_Compiler is used to create the customized IC_Manager. This customized IC_Manager explained in Section 4 of Chapter 8 becomes the CGI program to be invoked when the user clicks on the Web pages.

Then, you build the HTML pages for the active index system for distance learning you did for exercise #2. The end result should be a demonstration. You only need to e-mail your URL to the instructor so that the instructor can try your demo.

After that you can set up your working directory, by creating two subdirectories called "TAOML" and "source", and then copying all the files from the three directories IC_Compiler, IC_Manager and IC_Taoml to this "source" directory. Then you can follow the steps spelled out in MICE Application Development Steps in Section 6 of Chapter 8.

All the necessary files are available under ~jung/public/html in the following four directories: IC_Builder (the files you need to run the IC_Builder on PC), IC_Compiler (the files to run the IC_Compiler), IC_Manager (the files needed to compile the IC_Manager) and IC_Taoml (the interpretor to translate .taoml pages to .html pages) and can be downloaded.

EXERCISE 5:

After you have studied the various approaches, write a critique of these approaches, and a proposal of what you intend to do as a project. The critique and proposal should be between 1,500 and 2,000 words, with no less than 5 and no more than 10 references. A "template" is provided:

1st paragraph: Introduction.

2nd paragraph: My critique of various approaches.
(this could be the longest paragraph)
3rd paragraph: What I propose to do and how.
4th paragraph: Why this project is worth doing.
5th paragraph: Discussions.
5 to 10 references.

The critique/proposal should be e-mailed to the instructor.

PROJECT SUGGESTIONS:

Select an application area of interest to you. Design a multimedia application according to the suggested framework:

1. Syntax: Design the web pages and multimedia interaction languages.
2. Semantics: Design the actions associated with an application, and develop the active index cells (or agents). Associate the syntax with the semantics to organize the tele-action objects.
3. Pragmatics: Prototype the application using the MICE prototyping and multimedia information custom engineering environment.
4. Critique: Evaluate your design and suggest different design alternatives.

Those who are more interested in theory can investigate the formal specification of a multimedia application, and the verification and validation of such a formal specification.

References

[ACM94] ACM, Special Issue on Intelligent Agents, Communications of the ACM, Vol. 37, No. 7., July 1994.

[Aho86] A. V. Aho, R. Sethi, J.D. Ullman, *Compilers - principles, techniques and tools,* Addison-Wesley Publishing Co., 1986.

[Allen91] Allen J.F., Time and time again: The many ways to represent time, International Journal of Intelligent Systems, vol. 9, No. 3, June 1991, pp. 413-428.

[Arndt97a] T. Arndt, A. Cafiero, A. Guercio, "Multimedia Languages for Teleaction Objects", *Proceedings of 1997 IEEE Symposium on Visual Languages,* pp. 318-327, September 1997.

[Arndt97b] T. Arndt, A. Cafiero, A. Guercio, "Symbol Relation Grammars for Teleaction Objects", Technical Report, Dipartimento di Informatica ed Applicazioni, University of Salerno, 1997.

[Berners-Lee92] T. Berners-Lee, R. Calliau, J. F. Groff and B. Pollermann, "World-Wide Web: the Information Universe", Electronic Networking, Vol. 2, No. 1, Spring 1992, pp. 52-58.

[Berra90] P. B. Berra, C. Y. R. Chen, A. Ghafoor, C. C. Lin, T. D. C. Little and D. Shin, "Architecture for Distributed Multimedia Database Systems", Computer Communications, Vol. 13, No. 4, May 1990, pp. 217-231.

[Blak98] Micheal Blakeley, "ActiveWorks Focus on Business Practices", PC Week, Dec 28, 1998, p.37. [Allen83] Allen, J. F., "Maintaining Knowledge about Temporal Intervals," Communications of the ACM, vol. 26, no. 11, pp. 832-843, November 1983.

[Borenstin92] N. S. Borenstin, "Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work", CSCW 92 Proceedings, November 1992, pp. 67-74.

[Botto96] P. Bottoni, M. F. Costabile, S. Levialdi, P. Mussio, "Specification of visual languages as means for interaction", AVI '96 Int. Workshop on Theory of Visual Languages, 1996. URL: http://www.cs.monash.edu.au/~berndm/TVL96/tvl96-home.html.

[Brand88] F. J. Brandenburg, "On polynomial time graph grammars", In: LNCS 294, pp 227-236, 1988.

[Bulte91] D. C. A. Bulterman, G. van Rossum, R. van Liere, "A structure for transportable, dynamic multimedia documents", in Proceedings of the Summer 1991 USENIX Conference, Nashville, TN., June 1991, pp 137-155, 1991.

[Campb94] A. Campbell, G. Coulson, D. Hutchison, "A Quality of Service Architecture", Computer Communication Review 24(2):6-27, 1994.

[Catar98] T. Catarci, S. K. Chang, W. Liu and G. Santucci, "A Light-Weight Web-At-a-Glance System for Intelligent Information Retrieval", Journal of Knowledge-Based Systems, Elsevier, Vol. 11, 115-124, 1998.

[ChangH95a] H. J. Chang and S. K. Chang, "A Fuzzy Relation Language for Multimedia Presentation Scheduling", Technical Report, Department of Computer Science, University of Pittsburgh, March 1995.

[ChangH95b] H.J. Chang, T.Y. Hou, A. Hsu, S.K. Chang, "The Management and Application of Tele-Action Objects", *ACM Multimedia Systems J.*, Vol. 3, No. 5-6, Springer Verlag, 1995, pp. 204-216.

[Chang87a] Chang, S. K., "Icon Semantics - A Formal Approach to Icon System Design," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 1, no. 1, pp. 103-120, 1987.

[Chang87b] S.K. Chang, G. Tortora, A. Guercio, Bing Yu "Icon Purity - Toward a Formal Theory of Icons", *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 1, No. 3&4, 1987, pp. 377-392.

[Chang89] Chang, S. K., M.J. Tauber, B. Yu, and J.S. Yu, "A Visual Language Compiler," IEEE Transactions on Software Engineering, vol. 5, no. 5, pp. 506-525, 1989.

[Chang90] Chang, S. K., "A Visual Language Compiler for Information Retrieval by Visual Reasoning," IEEE Transactions on Software Engineering, pp. 1136-1149, 1990.

[Chang91] Chang S. K., *Principles of Pictorial Information Systems Design*, Prentice-Hall, 1991.

[Chang92] S. K. Chang, T. Y. Hou and A. Hsu, "Smart Image Design for Large Image Databases", Journal of Visual Languages and Computing, Vol. 3, No. 4, December 1992, pp. 323-342.

[Chang94a] Chang, S. K., M. F. Costabile, and S. Levialdi, "Reality Bites - Progressive Querying and Result Visualization in Logical and VR Spaces," Proc. of IEEE Symposium on Visual Languages, pp. 100-109, St. Louis, October 1994.

[Chang94b] Chang, S. K., S. Orefice, M. Tucci, and G. Polese, "A Methodology and Interactive Environment for Iconic Language Design," International Journal of Human-Computer Studies, vol. 41, pp. 683-716, 1994.

[Chang95a] Chang, S. K., "Towards a Theory of Active Index," Journal of Visual Languages and Computing, Vol. 6, No. 1, pp. 101-118, March 1995.

[Chang95b] Chang, S. K., G. Costagliola, G Pacini, M. Tucci, G. Tortora, B. Yu, and J. S. Yu, "Visual Language System for User Interfaces," IEEE Software, pp. 33-44, March 1995.

[Chang96a] S.K. Chang, "Extending Visual Languages for Multimedia", *IEEE Multimedia*, Vol. 3, No. 3, 1996, pp. 18-26.

[Chang96b] S.K. Chang "Active Index for Content-Based Medical Image Retrieval", *Journal of Computerized Medical Imaging and Graphics*, Special Issue on Medical Image Databases (S. Wong and H. K. Huang, eds.), Elsevier Science Ltd., Vol. 20, No. 4, 1996, pp. 219-229.

[Chang96c]S.K. Chang, P.W. Chen, G. Barry "A Smart WWW Page Model and its Application to On-Line Information Retrieval in Hyperspace", *Proc. of Pacific*

Workshop on Distributed Multimedia Systems, DMS'96, Hong Kong, June 27-28, 1996, pp. 220-227.

[Chang97] Chang S.K., Polese G., Thomas R., and Das S., A Visual Language for Authorization Modeling, Proc. of IEEE Symposium on Visual Languages, 1997, pp. 110-118.

[Chang98a] S. K. Chang, D. Graupe, K. Hasegawa and H. Kordylewski ,"An Active Multimedia Information System for Information Retrieval, Discovery and Fusion", IJSEKE, March 1998, Vol. 8, No. 1, 139-160.

[Chang98b] S. K. Chang, D. Graupe, K. Hasegawa and H. Kordylewski , "An Active Medical Information System using Active Index and Artificial Neural Network", in *Advances in Medical Image Databases*, (S. Wong, ed.), Kluwer, 1998, 225-249.

[Chang98c] S. K. Chang, E. Hassanein, C. Y. Hsieh, "A Multimedia Micro-University", IEEE Multimedia Magazine, Vol. 5, No. 3, July-September 1998, 60-68.

[Chen92] T. C. Chen, Y. Deng, S. K. Chang, "A Simulator for Distributed Systems Using G-Nets", Proceedings of Pittsburgh Simulation Conference, April 30 - May 1 1992, pp. 2705-2714, 1992.

[Chiu98] T. Chiueh and W. Wu, "Variorum: Multimedia-based Program Documentation System", Technical Report, CS Dept., SUNY, Stony Brook, 1998.

[Choi99] S. Y. Choi and A. B. Whiston, "The future of E-commerce-Integrate and Customize", Computer, Jan 1999, Vol. 32, No. 1, pp. 133-138.

[Colai94] F. Colaitis, "Opening Up Multimedia Object Exchange with MHEG", IEEE Multimedia 1(2):80-84, 1994.

[Costa95a] G. Costagliola, A. Guercio, G. Tortora, M. Tucci, "Linguaggi Visuali e Ingegneria del Software", *Procs. of AICA 95*, Chia (Italy), Sept.27-29, 1995, Vol.1, pp. 1-8.

[Costa95b] G. Costagliola, G. Tortora, S. Orefice, A. De Lucia, "Automatic generation of visual programming environments", Computer 28: 56-66, 1995.

[Costa97a] Costagliola G., De Lucia A., Orefice S., Tortora G., A Parsing Methodology for the Implementation of Visual Systems, IEEE Transactions on Software Engineering, vol. 23, n. 28, 1997.

[Costa97b] G. Costagliola, "Formal methods for visual editors and compilers", Technical Report, Dipartimento di Informatica ed Applicazioni, University of Salerno, 1997.

[Crimi90] Crimi, C., A. Guercio, G. Pacini, G. Tortora, and M. Tucci, "Automating Visual Language Generation," IEEE Transactions on Software Engineering, vol. 16, no. 10, pp. 1122-1135, October 1990.

[Deng90] Y. Deng, S. K. Chang, "A G-Net Model for Knowledge Representation and Reasoning", IEEE Transactions on Knowledge and Data Engineering 2(3): 295-310, 1990.

[Datto97] A. Dattolo and V. Lois, "Active Distributed Framework for Adaptive Hypermedia", International Journal of Human Computer Studies, 46(5):605-626, May 1997.

[Deng91] Y. Deng, S. K. Chang, "A Framework for the Modeling and Prototyping of Distributed Information Systems", International Journal of Software Engineering and Knowledge Engineering 1(3):203-226, 1991.

[Dimit94] D. A. Dimitroyannis, "Virtual Classroom: A Case Study", 1st International Conference on the World-Wide Web Proceedings, May 1994.

[Flet98] Tim Fletcher, Stephen G. MacDonell, William B. L. Wong, Early Experiences in Measuring Multimedia Systems Development Effort, 1998.

[Ferru94] F. Ferrucci, G. Tortora, M. Tucci, G. Vitiello, "A predictive parser for visual languages specified by relational grammars", Proc. IEEE Symp. on Visual Languages, pp. 245-252, 1994.

[Ferru96] F. Ferrucci, G. Pacini, G. Satta, M.I. Sessa, G. Tortora, M. Tucci, G. Vitiello, "Symbol Relation Grammars: A Formalism for Graphical Languages", *Information and Computation*, Vol. 131, No. 1, 1996, 1-46.

[Fujik95] K. Fujikawa, S. Shimojo, et. al. "Application Level QoS Modeling for a Distributed Multimedia System", Proceedings of 1995 Pacific Workshop on Distributed Multimedia Systems, Manoa, Hawaii, Mar. 31 - Apr. 2, pp 44-51, 1995.

[Garr97] A. Garrido, G. Rossi and D. Schwabe, "Pattern Systems for Hypermedia", Proc. of PloP'97, 1997.

[Goldberg92] Y. Goldberg, M. Safran and E. Shapiro, "Active Mail - A Framework for Implementing Groupware", CSCW 92 Proceedings, November 1992, pp. 75-83.

[Golin90] E. J. Golin, S. P. Reiss, "The specification of visual language syntax", J. Visual Languages and Computing 1:141-157, 1990.

[Grosk97] W. Grosky, R. Jain, R. Mehrotra, eds., *The handbook of multimedia information management*, Prentice-Hall, 1997.

[Guha95] A. Guha, A. Pavan, J. C. L. Liu, B. A. Roberts, "Controlling the process with distributed multimedia", IEEE Multimedia 2:20-29, 1995.

[Hala95] F. Halasz, "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems", CACM, Vol. 31, No. 7, 1995, pp. 836-855.

[Hardm94] L. Hardman, D. C. A. Bulterman, G. van Rossum, "The Amsterdam Hypermedia Model: extending hypertext to support real multimedia", Technical Report, CS-R9306, Centre for M&CS, Netherlands, 1994.

[Herman94] I. Herman, G. S. Carson, et. al, "PREMO: An ISO Standard for a Presentation Environment", Proceedings of the ACM, Multimedia '94 Conference, ed. D. Ferrari, San Francisco, October 1994.

[Hirak98] M. Hirakawa, Call for papers first international workshop on multimedia software engineering. URL: http://www.huis.hiroshima-u.ac.jp/~hirakawa/MSE98/mse98.html.

[Hira99] M. Hirakawa, "Do Software Engineers like Multimedia?" ICMCS99, Firenze, Italy, June 1999.

[Hoch98] T. Hochin, M. Harada, M. Nakata and T. Tsuji, "Virtual Multimedia Objects as the Platform of Customizing of Multimedia Data", Proc. Int'l Workshop on Multimedia Software Engineering, Kyoto, Japan, April 1998, pp. 20-27.

[Hou94] T. Y. Hou, A. Hsu, M. Y. Chiu, S. K. Chang and H. J. Chang, "An Active Multimedia System for Delayed Conferencing", Proceedings of SPIE, High Speed Networking and Multimedia Computing, February 1994, pp. 97-104.

[IOS94] International Organization for Standardization, Presentation Environment for Multimedia Objects (PREMO), ISO/IEC 14478-4.1, September 1994.

[ISO92] ISO 8879:1986, "Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)", ANSI, New York, 1992.

[Isak96] T. Isakowitz, E. A. Stohr and P. Balasubramanian, "RMM: A Methodology for Structured Hypermedia Design", CACM, Vol. 38, No. 8, Aug 1995, pp. 34-44.

[JBR95] Journal of Business Review, May 1995.

[Karsa95] G. Karsai, "A Configurable Visual Programming Environment", *Computer*, Vol. 28, No. 3, 1995, pp. 36-44.

[Khali96] Y. Khalifa, S. K. Chang and L. Comfort, "A Prototype Spatial-Temporal Reasoning System for Emergency Management", Proc. of International Conference on Visual Information Systems VISUAL96, February 5-7, 1996, Melbourne, Australia, 469-478.

[Li94] L. Li, A. Karmouch, N. D. Georganas, "Multimedia teleorchestra with independent sources: Part I - temporal modeling of collaborative multimedia scenarios", ACM/Springer-Varlag Journal Multimedia Systems 2(1):143-153, 1994.

[Lin94] C. C. Lin, C. S. Kao, W. C. Shang and S. K. Chang, "The Transformation from Multimedia Data Schema to Multimedia Communications Schema in Distributed Multimedia Systems", Proc. of Pacific Workshop on Distributed Multimedia Systems, Feb 26, 1994, pp. 1-13.

[Lin95] C. C. Lin, C. S. Kao, S. K. Chang SK, "Transformation among Multimedia Schemas in Distributed Multimedia Systems", Proceedings SPIE Multimedia Computing and Networking 1995, San Jose, CA., Feb. 6-8 1995.

[Lin96] Lin, C. C., J. X. Xiang, and S. K. Chang, "Transformation and Exchange of Multimedia Objects in Distributed Multimedia Systems," ACM Multimedia Systems Journal, vol. 4, no. 1, pp. 2-29, Springer Verlag, 1996.

[Little90a] T. D. C. Little and A. Ghafoor, "Multimedia Object Models for Synchronization and Databases", Proc. 6th Data Engineering Conference, Los Angeles, CA, February 1990, pp. 20-27.

[little90b] T. D. C. Little, A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects", IEEE JSAC 8(3) 413-427, 1990.

[Little91] T. D. C. Little, A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services", IEEE JSAC 9(9):1368-1381, 1991.

[Little93a] T. D. C. Little, "A Framework for Synchronous Delivery of Time-Dependent Multimedia Data", ACM/Springer Multimedia Systems 1(2):87-94, 1993.

[Little93b] T. D. C. Little, A. Ghafoor, "Interval-Based Temporal Models for Time-Dependent Multimedia Data", IEEE Transactions on Data and Knowledge Engineering 5(4):551-563, 1993.

[Lyar98] D. Lyardet, G. Rossi, D. Schwabe, "Using Design Patterns in Educational Multimedia Applications", Proc. of ED-MEDIA'98, Friburg, June 1998.

[MacDo98] Stephen G. MacDonell and Tim Fletcher, "Industry Practices in Project Management", Proc. of the Metric'98, IEEE CS press.

[Marri96] K. Marriot, B. Meyer, "Towards a hierarchy of visual languages", Proc. IEEE Symp. on Visual Languages, pp 196-203, 1996.

[Merce93] C. Mercer, S. Savage, H. Tokuda, "Processor Capacity Reserves for Multimedia Operating Systems", Technical Report, CMU-CS-93-157, School of Computer Science, Carnegie Mellon University, 1993.

[Murr98] P. Murray, "Can small companies afford to manage their knowledge?" Knowledge Management Associates, 1998. URL: http:// www.knowledge-at-work.com/ km_small_companies1.htm

[Nappi98] M. Nappi., G. Polese, G. Tortora, "Fractal Indexing and Retrieval System", *Image Vision and Computing*, 16, pp. 1019-1031, 1998.

[Neil98] S. Neil, "Delivering on Web Time", PC Week, Dec 28, 1998, p.104.

[Patr99] C. Partridge, "Embedded Wireless Connects Net to all and all to Net", IEEE Spectrum, Jan. 1999, p.38.

[Poles98] Polese G., MMDMS:A Framework for the Design of Spatial-Temporal Visual Languages, Ph.D. Thesis, University of Salerno, Italy, 1998.

[Ramae92] J. Ramaekers, G. Ventre, "Quality-of-Service Negotiation in a Real-Time Communication Network", Technical Report, TR-92-023, International Computer Science Institute, Berkeley, 1992.

[Reisi85] W. Reisig, *Petri Nets - An Introduction,* Berlin, Germany, Springer-Verlag, 1985.

[Saied96] H. Saiedian, "An invitation to formal methods", Computer 29:16-17, 1996.

[Smith92] J. R. Smith, "Columbia University's Imail (A Multimedia Mail Utility), Imail Operator's Manual", Image Lab - Imail Project, Center of Telecommunication Research, Columbia University, 1992.

[Son93] S. H. Son and N. Agarwal, "Synchronization of Temporal Constructs in Distributed Multimedia Systems with Controlled Accuracy", Technical Report, CS&IPC, University of Virginia, 1993.

[Staeh94] R. Staehli, J. Walpole, D. Maier, "Quality of Service Specification for Multimedia Presentations", Technical Report CS/E 94-033, Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, 1994.

[Stein90] R. Steinmetz, "Synchronization Properties in Multimedia Systems", IEEE JSAC 8(3):401-412, 1990.

[Tsai99] J. P. Tsai, "Knowledge-based Software Architecture", IEEE Trans. on Knowledge and Data Engineering, Vol. 11, No. 1, Jan 1999.

[Tzou87] K. H. Tzou, "Progressive Image Transmission: A Review and Comparison of Techniques", Optical Engineering 26(7):581-589, 1987.

[USWEB98] USWEB, Strategies for Growing Your Business through E-commerce, 1998. URL: http://www.usweb.com/services/ssc/res_lib/e_commerce.html.

[W3C98] W3C Recommendation, "Extensible Markup Language (XML) 1.0", 10 February 1998, URL: http://www.w3c.org/TR/REC-xml.

[Walla91] G. K. Wallace, "The JPEG Still Picture Compression Standard", CACM 34(4):30-44, 1991.

[Weitz94] L. Weitzman, K. Wittenburg, "Automatic Presentation of Multimedia Documents Using Relation Grammars", Proceedings of ACM Multimedia 94, ACM Press, New York, pp. 443-451, 1994.

[Weitz96a] L. Weitzman, K. Wittenburg, "Grammar-based articulation for multimedia document design", ACM Multimedia Systems J 4:99-111, 1996.

[Weitz96b] L. Weitzman, K. Wittenburg, "Relational grammars: theory and practice in a visual languages interface for process modeling", AVI '96 Int. Workshop on Theory of Visual Languages, 1996. URL: http://www.cs.monash.edu.au/~berndm/TVL96/tvl96-home.html.

[Wild91] J. C. Wild, K. Maly and L. F. Liu, "Decision-based Software Development", Journal of Software Maintenance, Vol. 3, No. 1, 1991.

[Wild98] J. C. Wild et al., "Project Management using Hypermedia CASE Tools", Technical Report, Dept. of Computer Science, Old Dominion University, 1998.

[Witte92] K. Wittenburg, "Earley-style parsing for relational grammars", Proceedings of IEEE Workshop on Visual Languages and Computing, pp 192-199, 1992.

[Wong97] S. T. C. Wong, H. K. Huang, "Networked multimedia for medical imaging". IEEE Multimedia 4:24-35, 1997.

[Woolf95] B. P. Woolf, W. Hall, "Multimedia pedagogues: interactive systems for teaching and learning", Computer 28:74-82, 1995.

[Xiang95] J. Xiang, H. J. Chang, C. S. Kao, S. K. Chang, "An Object Exchange Manager for a Distributed Multimedia Information System", Proceedings SPIE Multimedia Computing and Networking, San Jose, CA., Feb. 6-8 1995.

[Znati93] T. F. Znati, Y. Deng, B. Field and S. K. Chang, "Multi-Level Specification and Protocol Simulation for Distributed Multimedia Communication", Special Issue of International Journal in Computer Simulation on Simulation of Communication Systems, Vol. 3, No. 4, pp. 355-382, 1993.

Index