INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Université d'Ottawa · University of Ottawa

THE UNIVERSITY OF OTTAWA

FACILITATING INTELLIGENT MEDIA SPACE COLLABORATION VIA RASCAL – THE REFLECTIVELY ADAPTIVE SYNCHRONOUS COORDINATION ARCHITECTURAL FRAMEWORK

by

Robert Wayne Robbins, B.Sc., M.Sc.

A Thesis Submitted to the
Faculty of Graduate and Postdoctoral Studies
In Partial Fulfillment of the Requirements for the Degree of

Doctorate of Philosophy
in
Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
Electrical and Computer Engineering
University of Ottawa
Ottawa, Ontario, Canada

DEDICATION

This work is dedicated to my family – Mom, Dad, Grandma and aunts Jane and Marie – who were always a source of love, understanding and support throughout. Even though far from home, I knew they cared for me in ways nobody else could... and that meant more than words (even nearly 250 pages worth) can say.

ACKNOWLEDGEMENTS

By the time you get to write this section in a doctoral dissertation, there are just too many people to thank for having helped you over the years. So for those I forget to mention, know inside that you've made your mark in ways that will be forever remembered.

First, I would like to extend my gratitude to my supervisor, Dr. Nicolas D. Georganas, for his guidance, support and assistance in numerous and diverse ways over my years of study in Ottawa. Thanks also goes to my advisory committee, Dr. Dorina Petriu and Dr. Dan Ionescu, who also were examiners for my final defence along with Dr. Voicu Groza and Dr. Thiruvengadam Radhakrishnan. Your time and efforts were most appreciated.

Many others at the University of Ottawa have also impacted both me and my work over the years. Thanks goes to other students, past and present, in the Multimedia Communications Research Laboratory for their support in various ways. Special thanks goes to Nawel Chefaï, who has been a tried and true friend, confidant, colleague and a source of a special "je ne sais quoi". Many thanks also goes to our system administrator and lab manager, François Malric, who has helped make the lab an increasingly workable place. His good natured assistance to problems and issues both inside and outside the lab were appreciated more than he knows. Thanks to secretary/administrative assistant Michèle Roy for her buoyant attitude and peculiar brand of stubborn yet cheery wisdom (even if you say you like me because I'm "a character"...). A special word of appreciation goes to Dr. Moshe Krieger, for whom I was a teaching assistant for several years while at the University of Ottawa. His advice, encouragement and confidence in me both academically, pedagogically and personally meant a lot to keeping me going when I often thought of giving up.

The past few years have seen a lot happening for me in terms of personal growth and change. A big part of this has been my unique and interesting group of friends – of all kinds and from all walks of life. A special word of appreciation to Alain, André, James, Kevin, Nawel, Paul, Richard, Stéphane and Voja (in *alphabetical* order) for your support, advice, caring, and strangely enough, even your sometimes harsh but necessary moments. Kudos also go to Alain and Nawel for their valuable assistance as proof readers.

A long overdue thank you also goes to a unique and inspiring set of educators who motivated me throughout the years. Many thanks to Mr. Hilliard Sawchuk, Mrs. Pat Heuchert and Dr. Gerald Dueck for being the kind of educators that still continue teaching me things, even to this day.

And a final word of gratitude to my family, for their love and support. Special thanks to Jane for being one of my staunchest supporters as well as my biggest hero.

ABSTRACT

The increasing connectivity across domestic, commercial and recreational domains has lead to convergence not only in terms of technology, but also in terms of utility and expectation: people wish to work, play and interact together regardless of technological differences and without the need to continuously adapt to the idiosyncrasies of their collaborative tools. This desire to work together across diversity in location, technology, focus and intent creates the need to facilitate systems that are inherently malleable and that not only emphasize system function, but also how the system behaves. Combined with the proliferation and integration of multimedia technologies, these aspects have provided the impetus to realize collaborative multimedia systems that can meet the diverse and abstract needs of the collaborative process as its participants, scenarios and technologies change and evolve.

To do so, this work proposes a reflectively adaptive software engineering framework and support architecture to facilitate the construction of collaborative media spaces. Known as RASCAL, the Reflectively Adaptive Synchronous Coordination Architectural framework explicitly separates behaviour from functionality through the use of a meta-level architecture. Such an approach enables the potential development of intelligent, self-adaptive collaborative systems that can modify their internal construction to meet varying user expectations, behavioural patterns and physical resource constraints.

The general framework and supporting communication topology are presented along with an overview of a conceptual prototype called MSpace. Potential usage media-based scenarios ranging from business to artistic venues are explored along with a general evaluation of the system in terms of its reflective nature as well as its potential as a collaborative environment. Future work and additional research opportunities conclude with the observation that RASCAL's underlying essence of reflective adaptivity offers the most potential to meet the evolving needs, expectations and complexity that will someday make such an approach to system design not a luxury but a necessity.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

3D	Three Dimensional
A/V	Audio/Video
AI	Artificial Intelligence
ALIVE	Artificial Life Interactive Virtual Environment
AM	Adaptation Management
ATL	Active Template Library
BC	Behavioural Cascade
CA	Collaborative Association
CAAD	Collaborative Association Allocation Diagram
CAMP	Collaborative Association Membership Policy
CAPRE	Collaborative Association Pattern Reference
CARE	Collaborative Association Reified Elements
CASE	Computer Assisted Software Engineering
CBT	Collaborative Behaviour Tree
CCS	Coordinated Communications Subsystem
CCSfe	Coordinated Communication Subsystem Front End
CDE	Collaborative Development Environment
CF	Collaborative Focus
CLOS	Common Lisp Object System
CMSS	Collaborative Multimedia Support System
CO	Collaborative Object
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CORE	Collaborative Object Reified Elements
COSE	Collaborative Session
CS	Collaborative Session

CSCW ..... Computer Supported Cooperative Work

CSE ..... Collaborative Simulation Environment

CSIG ..... Collaborative Scenario Illustration Grid

CVE ..... Collaborative Virtual Environment

DCOM ..... Distributed Component Object Model

EM ..... Evolution Management

FBI ..... Functional Base Interface

FSM ..... Finite State Machine

GOPI ..... Generic Object Platform Infrastructure

HCI ..... Human Computer Interaction

ISO ..... International Standards Organization

JAMM ..... Java Applets Made Multi-user

JETS ..... Java-Enabled Telecollaboration System

JPEG ..... Joint Photographers Experts Group

LDU ..... Logical Data Unit

MFC ..... Microsoft Foundation Classes

MHEG ..... Multimedia Hypermedia Experts Group

MME ..... Multimodal Environment

MOM ..... Message-Oriented Middleware

MOP ..... Meta Object Protocol

MPEG ..... Motion Pictures Experts Group

OLE ..... Object Linking and Embedding

ORB ..... Object Request Broker

OS ..... Operating System

QoI ..... Quality of Interaction

QoS ..... Quality of Service

RASCAL ..... Reflectively Adaptive Synchronous Coordination Architecture

REXS ..... Reflective Execution Specification

RMI ..... Reflective Meta Interface

ROOM ..... Real-Time Object Oriented Modelling

RPC ..... Remote Procedure Call

RTP ..... Real-Time Protocol

SMPTE ..... Society of Motion Picture and Television Engineers

SPLINE ..... Scalable Platform for Large Interactive Networked Environments

TPM ..... Transaction Processing Monitors

UI ..... User Interface

UML ..... Unified Modelling Language

VR ..... Virtual Reality

1 CHAPTER ONE

INTRODUCTION

"The medium is the message"
Marshall McLuhan, 1967

When world-renown Canadian communications theorist Marshall McLuhan (1911 – 1980) first uttered the phrase "the medium is the message", he exemplified a critical aspect of the modern digital communication revolution well in advance of its arrival. Unlike many, McLuhan suggested that communication was more than just a single dimension; rather, he offered that the means and manner by which information is delivered plays a significant role in its understanding. And nothing could be further from the truth when utilizing media not only to communicate but also to collaborate.

As part of the modern-day realization of McLuhan's 1962 prediction of a "global village", there is an increasing effort towards enabling people to work and interact together across distances and despite differences in locality. Research and technological support towards this end are classified under the notion of *computer supported collaborative work* (or CSCW). The aim of CSCW systems is to increasingly utilize the necessary means of communication and interaction to allow their users to perform their tasks in ways that are familiar, appropriate and useful. Consequently, there is an increased tendency towards the use of the diverse range of informational representations known as *multimedia* and familiar user-level computational tools, known as *applications*. This fusion of multimedia and distributed interaction within a logically shared environment forms the foundation of a new meta-medium: the *collaborative media space*.

1.1 Motivation

A collaborative media space can best be regarded as a logical CSCW abstraction that enables collaboration through the use of multimedia. As such, it blends the concepts of collaboration and media-based environments in a synergistic manner. The realization of this synergy results as much from the individual application of these concepts as well as how they work together.

Collaboration is a ubiquitous yet somewhat esoteric process. Defined as the ability "to work jointly with others" and "to cooperate with an agency or instrumentality with which one is not immediately connected" [Web89a], collaboration permits people to jointly explore ideas toward whatever purpose and whatever end is desired. It is a process which enables the cooperative investigation and/or development of ideas; that is, it is the notion of people working together to accomplish a goal. Thus collaboration can be seen as an integration of people applying tools and methodologies to achieve a given purpose. The familiar notion of collaboration is a room full of people working together, helping each other out with a particular project. Such a *co-located* scenario is quite common in terms of business meetings, research laboratories and even school classrooms.

Increasingly, however, there is a desire to collaborate without the need to be in the same place at the same time; that is, people want to work together at a distance. Consequently, means by which collaborators can achieve a useful level of interaction across distances need to be addressed. Hence, CSCW systems that facilitate this are increasingly in demand, and one such approach is the utilization of multimedia.

The role of conventional multimedia systems has been the dissemination of information. Whether textual, graphical, auditory or visual, such systems were designed to augment the *presentation* of ideas by representing them using the most appropriate format. *Collaborative multimedia*, however, goes beyond the role of exposition to provide support for distributed groups. By integrating multimedia technology within CSCW environments, media are not the sole focus of the system but a natural and intuitive means to facilitate group interaction and the collaborative exchange of ideas.

The classical model of collaborative computing is that offered by teleconferencing. While such a model can be classified as multimedia , its collaborative potential falls short because it does not support the range of media which allow users to mimic traditional shared work spaces. In real-world collaborative exchanges, people see and talk to each other, manipulate and point at shared props, sketch ideas on shared surfaces and view other media (such as film clips) in a mutually common environment. In short, participants and the objects they use form a shared "media space" in which users interact with each other through the experience and manipulation of multiple media. The expressiveness of a diverse media palette combined with the distributed work environments goes beyond the utility of teleconferencing through the blending of *presentational*, *conversational* and *interactive* multimedia. For example, the ability to integrate dynamic, complex and real-time problem domains and techniques such as scientific visualization, computer-assisted design (CAD) and distributed musical composition brings both new opportunities and increased flexibility to the collaborative process.

Historically, the majority of media-related topics have been grounded in the artistic arena. However, with the advent of capable technology, research and development of media-based systems within the scientific and engineering communities have been popular in recent years. Despite their common underpinnings, however, this latter work has generally been devoid of artistic reference. While primarily due to the different focus between the two schools of thought, the "real world" essence of a "media space" is quite well shown through their combination. Given the word "space" typically suggests a physical manifestation, consider an example of an actual physical media space, which is located outside the "new" Ottawa city hall (formerly the Regional Municipality of Ottawa-Carleton complex). As part of the complex's landscaping, an artistic construction known as "VIP – Virtual Instrument Paradigm" was commissioned from local artist Michael Bussiere to provide an example of "interactive art". Shown in Figure 1.1, the construction is built into the landscape by integrating itself into the footpath in front of the complex. As users walk along the path, they interleave amongst a set of statues which generate musical sequences depending on their movement. The use of motion sensors in the hook-like top of the statues provide real-time input to the backend processing system which composes complex musical sequences from pre-recorded riffs relative to the particular statue and the kinds of movement made within its sensitivity area. Consequently, the pattern of movement along the path has a direct

effect on the music generated by the media space. The independence of the statues provides the ability of the space to accept input from multiple sensors at once and consequently multiple people can interact along with the path to create complex musical constructions. Therefore, in a very general sense, the (media) space provides the ability for people to loosely collaborate by coordinating their movements within it, relative to each other and the sensitivity areas of the statues.

Taken logically, however, the potential of a media space is greatly enchanced given that its elements can be geographically distinct and manipulated remotely through the use of networked communications facilities. As such, the term "media space" originated from work at Xerox Parc [Stu86] [HM89] in the mid to late 80's and one of the more broad discussions of its usage [BHI93] addressed aspects of group interaction between co-workers at remote sites in both social and work settings. Using a combination of digital and analogue technologies, initial media spaces connected both individuals and groups using an analog audio/video network in parallel with a network of workstations. In contrast, our interpretation of a media space is based on the premise that all media space elements are digital and users interact using a unified, more closely integrated "point of presence" within the space. Collaboration is facilitated via whatever *objects* (media, applications and so forth) are shared between users and how they are manipulated. Indeed various media and interfaces (including user interface peripheral devices such as video capture/display) are integrated into the space as objects rather than separate parallel systems.

Accordingly, media spaces can be defined as "large, shared, interactive, media-rich environments that combine graphics, video, sound, speech, images, and networking to create compelling interfaces for learning, entertainment, social interaction, and business" [IEE96]. The notion is that of a general-purpose multimedia-enabled distributed real-time synchronously shared environment. As such, the media space logically groups collaborators and their information (i.e. media) as well as the applications used as part of their work. In this way, a media space is inherently a collaborative facility that can enable people to interact across distances through the manipulation of media and application tools within a virtual environment. That is, users interact through cooperative and collaborative exchanges via the shared experience and manipulation of a media space, in which media are not the focus but the facilitator. The space itself forms a medium that is not only its media

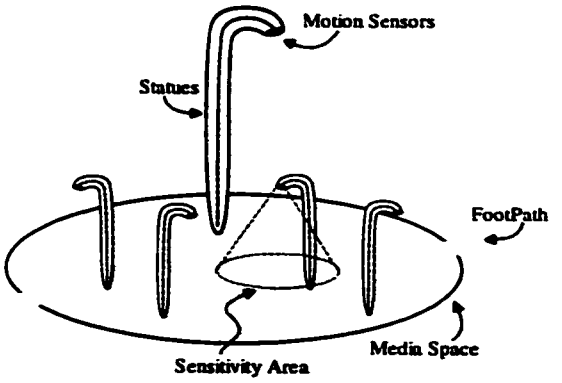


Figure 1.1: VIP – A Real Life Media Space

content but also the composition of those elements. Being more than the sum of its parts, the relationships amongst the space's constituent elements as well as their individual semantics constitute the meaning of the space, facilitating its usefulness and the ability of its users to understand and interact with each other. That is, there is a need for *semantic coherence* which is based on ensuring the very often subtle and implicit *coordination* of activities that are part of the collaborative process [GG95].

1.2 Focus, Objectives and Scope

Akin to facilitating the collaborative process in the real world, a CSCW framework must unify and enable synergy between its components in a natural, utilitarian and conducive manner. It must provide means to specify, utilize, adapt and evolve such elements as they become relevant to the collaborative process.

The focus of this work is the realization of a model for the design, construction, management and evolution of collaborative media spaces. These media spaces provide for collaboration utilizing any combination of presentational, conversational and interactive multimedia, with no fixed user interface, application toolset, behavioural model or policy. The model offers a framework and methodology for building systems which integrate behavioural management facilities through the use of an intelligent policy-driven communication subsystem. Rather than address specific low-level protocol or algorithmic approaches for data transfer, synchronization or the like, this work focuses on how to integrate media space elements in a way that allows run-time evaluation, manipulation, control and evolution of the space and its elements both in terms of structure as well as their behaviour. The goal is not to design a specific application or system with an intrinsic but hard to qualify behaviour. Instead, the goal is to design a way to build systems out of specific functional and behavioural entities. The intent is that the system can monitor, adapt, modify and evolve both function and behaviour to meet its changing needs over its lifetime. Our approach provides an integrated solution so that these different aspects are sufficiently modular but not disjoint.

The term *behavioural management* is used to address the needs of both the real-time activities of collaborating users as well as the algorithmic performance of computational entities within the space. In general, behavioural management addresses the coordination of entities, with attention to the highly dynamic and special needs of media synchronization, given the potential heterogeneity of media and activities within the space. The fundamental requirement for collaborative success is being able to understand the actions performed by the collaborators. That is, the collaboration must make sense to those involved in the collaborative process and offer semantic coherence by accurately representing the entities and actions in the space, both as they perform their specific function and as they relate to each other throughout their interaction. To lend meaning and utility to the collaborators' behaviours, their actions and the relationships between them must occur in an organized and coherent manner. Therefore, the coordinated management of such behaviours becomes a fundamental issue in providing a meaningful collaborative environment.

Collaborative systems based on the media space approach can be seen as part of realizing flexible, dynamic and useful collaborative environments. The combination of computer technology, human and social interaction, as

well as different media, create a spectrum of possible behaviours, functionalities and requirements. In addition, being able to vary these aspects both within and between collaborative sessions, amongst individual collaborators and their applications further exacerbates the complexity of supporting the collaborative process. For example, participants may work alone or in groups; they may use commonly shared or their own private application tools. Therefore, the potential heterogeneity of activity within a media space suggests that individual applications would benefit from a separate facility that is responsible for ensuring a coordinated and semantically coherent workspace. By doing so, the internal complexity of applications could be reduced and their potential flexibility increased through separating the issues of "what" the application does vs. "how" it is done relative to a particular configuration or usage of the media space. This separation of concerns is beneficial because it:

- abstracts the role of coordination within distributed environments, which can be difficult as well as system dependent
- lessens processing and resource management requirements within an application
- reduces design and implementation complexity of applications as well as their possible hardware and support software requirements
- promotes increased flexibility, upgradeability and policy malleability by reducing said complexity
- increases potential of the application because focus is on "what" vs. "how"

This coordination [MC94] is necessary in order to achieve an environment in which the actions performed upon objects and the interactions between them make sense; that is, meaning is maintained through synchronizing participants and their actions. While multimedia synchronization has usually referred only to the temporal relationships among media [SN95], this is insufficient within collaborative systems. Participant interaction and media processing must be integrated so as to provide a naturally fluid, predictable and reasonable work environment. Every activity must be coordinated so as to ensure that it maintains its own semantics (such as timeliness), while interacting activities must also occur in the correct temporal order (i.e., precedence relationship) to ensure causal dependencies of collaborative actions are reflected and maintained throughout the shared space [MC94] [Yav92] [SS94]. That is, collaborative systems must also provide for *causal synchronization* to ensure the legitimacy of participant interaction and object manipulation within a media space.

The relationship between disparate but collaborating entities within the space forms a collection of varying synchronization requirements and behaviours. Some entities behave in an interleaved fashion while others function in a true parallel manner. Therefore, the system forms a *multi-synchronous space* [Dou94] whose breadth of requirements cannot (in practical terms) be appreciated (or supported) by the coordination and synchronization logic of any one entity within it. Beyond the need for "lots of very fast data" is the need for real-time data, delivered and manipulated in an orderly and timely manner. Therefore, any collaborative multimedia system needs to address both temporal and causal synchronization as part of managing shared space

coordination in a fluid and integrated manner. Consequently, a more global perspective and an architecture which can address such issues is necessary.

Collaborative space research has traditionally revolved around elements of collaboration arising from process, sociological and technological perspectives: (1) specific application environments [AEH88] [Eri94]; (2) human-computer interaction [GG95] [LL90]; and (3) communication and/or application protocols [Yav92] [AY96]. In the first, collaboration is usually approached in terms of a specific program's functionality, while the second addresses how well the elements of a system and/or its user interface suit the needs of the collaboration and its participants. Consequently, coordination and synchronization are often either not addressed or embedded within a given application. Additionally, collaborative utility is limited by the use of specific tools in a particular context because the collaborative utility is "wrapped up" in the design and complexity of the chosen tool set. Alternatively, the protocol-based approach is more versatile and often addresses these issues; however, it is based on the notion that a set of data exchange rules is the predominant issue. Often synonymous with the notion of floor control [DG97], the protocol perspective yields "message sequence chart"-based solutions which range from the very general to the very specific and/or complex.

Similarly, multimedia research is often directed at specific niche areas, including: coding techniques, quality of service issues, synchronization and communication protocols. In addition, there is often a strong emphasis on the development of specific applications for specific usage scenarios. The result is that approaches are often divided into ways to support one of presentational, conversational or interactive multimedia. There is little emphasis on the integration of these techniques, which is often due to the emphasis on function versus behaviour within media based systems. The result of such emphasis is that the effects of implicit behaviours are hard to qualify and consequently, it is difficult to integrate them together in order to address variable and diverse behavioural requirements.

Accordingly, this work presents a reflective meta-level approach to collaborative media space management. Emphasis is given to the design of a versatile and enabling framework for building a broad range of media-based collaborative environments with specific support for diverse behavioural requirements. How to provide for these various facets in a flexible and adaptive fashion illustrates the importance of addressing how "the pieces of the puzzle" are put together. Hence, this work addresses a cross-section of issues related to multimedia and distributed media systems as well as collaboration and collaborative systems. In doing so, the scope and limits of this work are outlined below.

Specifically, this work addresses the need for a framework and methodology which be can used to build a versatile and enabling infrastructure for user-level collaborative environments. The proposed methodology does not preclude any of the above efforts but can be viewed as a potential way to organize, relate and integrate different approaches (including user interface technologies, applications and protocols) as required. The architecture's design, its underlying software engineering and the role of the communication infrastructure are therefore primary considerations and are based on a key underlying principle: management and domain functionality must be strictly separated to facilitate a flexible, modifiable and adaptive system. In particular, the

behaviours of both the environment and individual entities within it should be managed separately from their domain functionality. This work does not directly address specific communication issues such as network protocols or the like. The issues of coordination and synchronization are primarily directed at addressing high-level issues which enable participants within the collaborative process to function in a cohesive, coordinated and coherent manner. Because low-level media synchronization issues (such as pacing the playback of continuous media streams) have already been addressed by a large body of research (see [BS96] for an overview), they are considered secondary relative to the goals of this work. Consequently, they will be touched upon only to illustrate completeness and orthogonality of the model, how it facilitates collaboration by enabling a flexible media space and how it supports the integration of presentational, conversational and interactive multimedia in a unified framework.

To meet the overall goals and focus of this work, the following objectives have been set:

- Illustrate a media space as an integrated cooperative system involving the following elements:
  - *media* – the various informational formats which constitute multimedia, the different types, their requirements and so forth
  - *human participants* – which interact within the space by manipulating and experiencing the different media
  - *technological entities* – elements which work with or enable other media space entities; for example: applications, intelligent agents and so forth
- Specify a coordination-based media space architecture based on the use of a *coordinated communications subsystem* (CCS).
- Define the need, role, essence and elements of a *collaborative multimedia support system* (CMSS) relative to the CCS and the given media space architecture.
- Specify a meta-level behavioural management model utilizing the above media space architecture; this includes the role of:
  - *reflection* – originally introduced within the field of artificial intelligence by [Smi84] [Smi85]
  - *meta-technologies* – the notions of metamodels, metaprotocols and metacomputing
  - *intelligent network services* – as illustrated through the facilities offered via the CCS and the CMSS
- Explore the integration of presentational, conversational and interactive multimedia within a unified collaborative multimedia framework
  - This will, in part, address the nature and role of synchronization between live and synthetic streams and their integration within the "multi-synchronous" nature of a shared media space.

- Provide an integrated and unified approach to the above concerns and issues within a model called *RASCAL* – the *Reflectively Adaptive Synchronous Coordination Architectural framework*
    - Illustrate the principles of the RASCAL framework using a conceptual prototype called *MSpace*

1.3 Thesis Statement

Therefore, the thesis of this dissertation can be summarized as follows:

> A versatile, efficient and scalable real-time collaborative media space can be implemented based on an abstract model of functional and behavioural components, which explicitly considers the ability to evaluate, manipulate and control the execution and relationship between them in a reflective manner. Such an approach allows for and promotes heterogeneity and increased quality and flexibility in terms of user and computational behaviours within the space as it is used over time, for different contexts and given different underlying technologies.

1.4 Contributions

Given the overall thesis, focus and objectives of this research, this work addresses the need to provide for advanced features and reliable behaviours within multimedia collaborative environments not by the use of specific low-level approaches, but by the correct software engineering and system construction techniques. It takes a responsibility-driven methodology and combines facets of artificial intelligence, active (intelligent) networks, real-time systems, multimedia and collaborative environments.

The main contributions of this research are as follows:

- Development of a behavioural-based collaborative media space model using an intelligent communications subsystem. The subsystem is controlled by explicit and active behavioural policies rather than implicit behavioural implementations within functional entities.
- Use of reflection and meta-technologies within real-time collaborative systems:
    - As a comprehensive design methodology to support the coordination and synchronization of media space elements
        - Between live and synthetic streams
        - Providing an approach to both causal and temporal media synchronization
        - Facilitating the use of different synchronization models and paradigms
        - Facilitating the notion of "context-mediated behaviours" by the ability to dynamically replace and adapt behaviours throughout a collaborative scenario

    - As an enabler of a self-investigative and self-regulating approach to adaptive media space synchronization and coordination, called "Reflective Synchrony".
    - As a facilitator of a behavioural agent approach to active collaborative space management and adaptation;
- Development of the RASCAL framework, a meta-level framework for the construction of collaborative media space environments:
    - This forms the basis of a general design methodology for complex real-time collaborative multimedia systems.

1.5 Thesis Organization

The rest of this document is organized as follows: Chapter 2 overviews the relevant literature of interest to this research while Chapter 3 presents the general principles and architectural considerations of the RASCAL framework. Chapter 4 and 5 discuss the framework in terms of example scenarios and its illustration by means of a conceptual prototype. Chapter 6 then addresses issues of performance, viability and related issues while Chapter 7 summarizes and concludes.

1.6 Publications Arising From This Research

Throughout the course of this research, the following related publications were produced:

- *Reflective Media Space Management Using RASCAL.* In Proceedings of Reflection'99: 2nd International Conference on Metalevel Architectures and Reflection. July 19 - 21, 1999. St. Malo, France.
- *Reflective Pattern-Based Approach to Media Space Management.* In Proceedings of MMCN'99: Multimedia Computing and Networking 1999 (Electronic Imaging/Photonics West). January 25 - 27, 1999. San Jose, CA.
- *Collaborative Media Space Architectures.* In Proceedings of CCBR'98: Canadian Conference on Broadband Research. June 21 - 23, 1998. Ottawa, ON.
- *Shared Media Space Coordination: Mixed Mode Synchrony in Collaborative Multimedia Environments.* In Proceedings of IEEE Multimedia Systems'97. June 5 - 9, 1997. Ottawa, ON.
- *Shared Media Space Coordination in Collaborative Multimedia Environments.* Poster for Canadian Institute for Telecommunications Research Conference. August 27 - 29, 1997. Toronto, ON.
- *Mixed Mode Synchrony: Integrating Live and Synthetic Media within a Shared Environment.* Poster for Canadian Institute for Telecommunications Research Conference. August 28 - 30, 1996. Montebello, QC.

2 CHAPTER TWO

CONCEPTUAL BACKGROUND OVERVIEW

This chapter overviews the background research and conceptual bases for the work presented in this dissertation. First, the areas of collaboration and multimedia, coordination and synchronization as well as the relevant aspects of systems engineering are discussed. This is followed by an outline of specific implementation technologies used in the MSpace prototype.

2.1 Conceptual Background

As part of understanding the intent and applicability of the RASCAL framework, three major areas of work are highlighted in relation to this work. In particular, this includes the areas of: (1) collaboration and multimedia; (2) coordination and synchronization; and (3) systems engineering.

2.1.1 Collaboration and Multimedia

This section presents issues and concepts related to the notion of collaboration and the role of media technologies in relation to the collaborative process.

2.1.1.1 Collaboration

As stated in Chapter One, collaboration is a familiar yet rather abstruse process in which people jointly explore ideas toward a specific purpose and desired end. Regarded as the "cornerstone of group activity" [EGR91], it embodies the cooperative investigation and development of ideas, and is the notion of people working together to accomplish a goal.

Traditionally, collaboration has been limited to co-located (i.e. physically shared) spaces which belie the many subtle and assumed aspects as to how collaboration is actually achieved. These details exact considerable and specific demands from those involved in the collaborative process; and in particular, successful collaboration must consider the context in which it is facilitated, those who are participating and their individual requirements. In short, the success of a collaboration depends on its participants and their behaviour within the process itself.

Additionally, such real-world collaboration is inherently a media-oriented activity: people see each other, talk to each other, point at shared props, sketch on shared surfaces and can view other media (such as video clips) in a mutually common environment. In short, as part of the collaborative effort, the participants and the objects they use literally form a *physically shared media space* in which users interact with each other through the experience and manipulation of media (such as books, diagrams, writing instruments, blackboards and so forth).

The collaborative process utilizes and benefits from the breadth of media and interaction styles used by the participants. In particular, by combining presentation (of material), conversation (between participants) and interaction (between participants with respect to objects in the shared space), the appropriate medium for the task at hand can be experienced in a shared and globally aware manner.

In order to support such interaction in a non-colocated environment, certain issues must be considered in the provision of the collaborative support system. For example, in order to provide for real-time interactive exchanges between users in a non-colocated environment, the issue of *awareness* becomes fundamentally important [GG95]. In a physically shared space, collaborators can physically observe, encompass and react to the entire space at a single instant. All participants can simultaneously evaluate their common environment and adapt to their circumstances, fluidly, seamlessly and coherently moving from one type of activity/behaviour to another as required by the situation at hand. Consequently, participants can be seen to be universally aware of the shared collaborative state. In a *logically* shared media space, however, collaborators do not implicitly have this simultaneous awareness; therefore, it must be provided as part of maintaining a globally coherent environment. This necessary acuity of media space state is an important element in maintaining an environment in which all participants can relate to activities at the same time. In order to interact in an equitable manner, activities must occur in a synchronous manner across all participants [AY96] so as to appropriately (and explicitly) notify participants of actions which affect them while also providing a non-obtrusive, up-to-date and timely group context [EGR91]. Activities need to occur in a timely and orderly manner and in relation to the overall goal and state of the collaboration; that is, activities must be coordinated and synchronized so as to provide a semantically coherent environment in which individual collaborators know what is going on and can interact in a legitimate and sensible fashion. While often provided by social convention and human judgment in a co-located encounter, these aspects must be pedantically accounted for in computer-based systems used to facilitate it. Therefore, synchronization is a fundamental aspect of a any system which supports maintaining a meaningful collaborative environment. This notion of coordinating media space activities is explored further in upcoming sections.

2.1.1.2 Collaborating Via Computer: Groupware and CSCW

As applied to collaboration, the study of office automation systems has defined two broad categories of cooperative computing: (1) *groupware*; and (2) *computer-supported collaborative work* (commonly known as CSCW). Defined as *"computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment"* [EGR91], the primary focus of groupware is to provide a group interface for a shared task. In a similar vein, computer-supported cooperative work systems deal with how technology (specifically computer systems) can assist in the work process. These two areas are often combined into a single focus which addresses how groups of people can work together in a logically shared environment with the assistance of computer support.

Groupware systems span a spectrum of usage models and the time/location matrix illustrated in Figure 2.1 is a well-known taxonomy [EGR91]. Systems which support synchronous activity are categorized as *real-time groupware* while those that support asynchronous activity (such as electronic mail) constitute the non-real-time category. Synchronous systems therefore offer a concurrent shared environment in which multiple users can interact simultaneously while asynchronous systems provide for serial non-shared interaction. As a rule, most groupware systems support interaction along only one such dimension; and similarly, RASCAL focuses on the provision of synchronous distributed interaction as applied to same time/different place collaboration.

Groupware systems have also been classified based on application functionality, including the categories of message systems, multi-user editors, computer conferencing, intelligent agents, group decision support and coordination systems. An overview and discussion can be found in [EGR91]. Systems often span multiple categories using this classification system and can be used in an overlapping manner by a group to accomplish its goal. For example, more than one type of desktop publishing tool may be required to produce a brochure. Consequently, collaboration can be seen as a blend of activities used in varying ways throughout the collaborative process and across those participating in it. Therefore, the utility of a collaborative environment requires an open and flexible system which can support the integration of different technologies to support the given collaborative task as its needs change and evolve over time.

To support the diversity associated with the notion collaboration, several different perspectives exist in the application of groupware technologies. These include: distributed systems, communications, human-computer interaction (HCI), artificial intelligence (AI) and social theory. Again, an overview can be found in [EGR91]. Most systems tend to support group work according to a particular approach and with their own unique focus. In contrast, RASCAL provides an integration framework for the inclusion of the appropriate elements within a collaborative space.

While predominantly blending a distributed system approach to media space management, RASCAL provides a uniform way to address the different facets of media and collaborative computing using a meta-level approach to defining, implementing and managing aspects of the system. This includes an adaptive meta-level behavioural management system that parallels the synchronously shared workspace while considering human-computer interaction and artificial intelligence perspectives through the potential to integrate intelligent self-regulating collaborative behaviours via reflection. These issues are further explained in upcoming chapters.

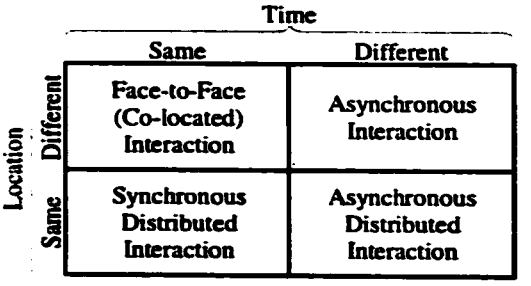


Figure 2.1: Groupware Time/Location matrix

2.1.1.3 Multimedia

In all facets of modern day society, the role of media-intensive communications is becoming increasingly predominant. While traditionally offering only text-based environments [LHG93] [LH88], computer systems now provide support for the different modes of expression used by people in their daily communications. This diversity in the representation of computer-based information forms the basis of multimedia. Defined as *"the integrated generation, representation, processing, storage and dissemination of independent machine processable information expressed in multiple time dependent and time independent media such as data, graphics, drawings, voice, audio and video"* [Ste90], multimedia matches the natural intent of human communication by employing flexible and meaningful representations of information relative to the subject at hand. Its use supports appropriate communication styles and provides a naturally expressive means for explaining and understanding complex subjects, issues and processes.

Based on the above definition, systems capable of creating, storing, retrieving, transporting and manipulating such information are called *multimedia systems* [Geo94]. The development and usage of such systems has traditionally been based on the modality of their media as well as their underlying interaction model. The result is three basic categories: presentational, conversational and interactive.

Presentational multimedia systems are those designed to playback pre-constructed multimedia scenarios (presentations containing synthetic media objects for display according to a pre-defined timeline) to a passive audience. This type of system is based on the traditional lecture style of delivery. As illustrated in the top two frames of Figure 2.2, the architecture of presentational systems utilizes the retrieval of pre-stored media from remote servers which are then displayed to the viewer. Such systems are typified by kiosk-style information booths like those used in educational [SKD96] and tourism sectors. In terms of media mode, presentational systems utilize synthetic media.

Conversational multimedia systems (middle frame of Figure 2.2) are those which allow people to engage in a dialogue using voice (and possibly video) in a real-time synchronous manner. Such systems can range from audio-enabled "phone" programs like InternetPhone [Voc97] to audio/video teleconferencing systems [EGR91]. Therefore, the most basic notion of conversational multimedia is represented by the standard telephone call while a more elaborate version would use an audio/visual system involving a video camera, microphone, television monitor and speaker system. In terms of media mode, conversational systems are based on live media rather than synthetic.

The notion of *interactive multimedia* augments the presentational classification by enabling participants to control and interact with their environment. Within a typical interactive system, users have control over a presentation's progress, such as being able to rewind, fast-forward or hyper-link to other parts of the presentation [SKD96]. The architecture, illustrated in the bottom frame of Figure 2.2, utilizes "control points" within the scenario's timeline to allow users to choose different possible playback options/sequences.

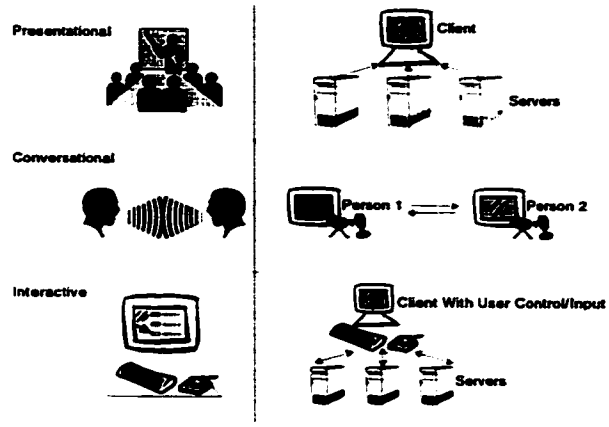


Figure 2.2: Traditional Multimedia Classifications

2.1.1.4 Multimedia System Design

The lineage of traditional multimedia systems is one of disseminating information. As such, different kinds of systems were built to present ideas using suitable mechanisms and particular formats, often being divided into the above classifications. To be effective across a wide range of activities, however, collaborative multimedia systems must support the integration of the three multimedia domains within a single unified abstraction.

Because multimedia-based collaborative environments are intended to offer some of the same functionality as legacy systems, they necessarily deal with some of the same issues. Furthermore, the integration of these different functionalities and characteristics need to be considered in how they impact upon a collaborative setting. These issues are now briefly overviewed relative to the definition of "multimedia" and as a means of highlighting differences between the various system categories.

Data Representation. While the obvious effect of data representation is the size of a media object, data representation is also important in that it characterizes the processing characteristics of specific media types and their modality[1]. Because the utility of a multimedia system is affected by its ability to handle a variety of data types, systems need to support a wide range of media (e.g. audio, video, imagery, etc.) as well as many of the encoding schemes that exist within each class. Since some schemes (such as MPEG [BL94]) are computationally intensive, data representation implicitly defines a system's processing requirements and directly impacts upon its real-time capabilities. The fluidity afforded conversational multimedia and real-time groupware, for example, can be heavily influenced by the performance of such techniques as utilized within the system. Non-real-time systems which have the ability to perform data pre-fetch and decode in advance are able to put more emphasis on the compression and data size issues while live media scenarios need to be more concerned with the real-time performance of their codecs.

---

[1] See Appendix A for a more detailed presentation of the nomenclature used with respect to media organization and temporal characteristics.

Authoring. The first step in using a presentational multimedia system is the creation of the presentation itself. This task, known as *authoring*, involves the specification of the presentation's media objects and their temporal relationships; that is, it is when the content and organization of the presentation is determined. Software packages that assist in this process are called *authoring packages* and typically offer a number of input metaphors such as scripting, flowcharts or card stacks which generate a timeline that schedules the rendering of each media object as the presentation is rendered to the user.

As an example, consider Little and Ghafoor's Timed Petri Net model [LG90] [LG91]. Based on a primitive unit known as a *temporal interval* (a non-zero period of time), the model specifies how a presentation (in which each media object corresponds to a temporal interval) is then represented as a petri net, which renders the presentation according to the "firing pattern" of the petri-net (Figure 2.3). Therefore, the inherent synchronization offered by the petri-net mechanism is inherited from its use as an authoring tool.

Another example is MHEG, an ISO standard which defines a system-independent representation for the structures used to store, exchange and execute interactive multimedia presentations [ME95]. The standard builds multimedia presentations out of "MHEG objects" which are defined to include content data, behaviour and composition. An object's content data represents its "computerized representation," behaviour represents its applicable actions and composition refers to how separate objects can be grouped together. Interactive presentations are defined by allowing a user to control presentation progress by manipulating specific objects within it.

In each of these two cases, the *a priori* existence of the media objects and their interactions allow the presentation and its temporal relationships to be specified before playback. In presentational systems which support user interaction, variation in playback corresponds to "controlling" an individual object's behaviour, such as fast-forwarding a video clip. Therefore, such interaction and its effects on playback (that is, the

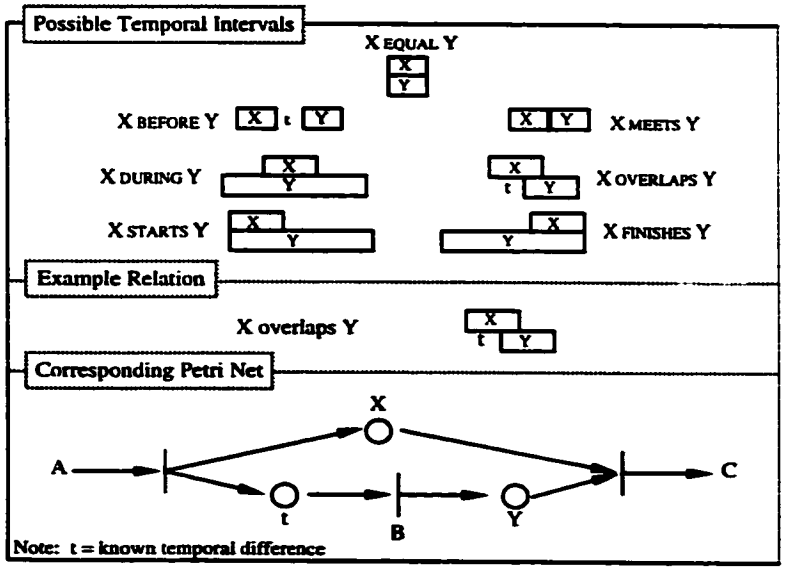


Figure 2.3: Various Multimedia Synchronization Specification Techniques

presentation's timeline and possible syntax and semantics) are known in advance and can accounted for in the specification.

Within a collaborative environment, however, "authoring" is not a single isolated step. Rather, it is an on-going process which occurs "on-the-fly" throughout the collaboration; that is, the collaboration is itself the authoring process in which the contents (i.e. the media, their characteristics, requirements and values) along with their interactions (i.e. their synchronization and coordination) change during use. In particular, unlike a presentational system, the existence and characteristics of the media objects and their interactions are not necessarily known *a priori*. Therefore, the media used within the collaborative effort, as well as their coordination and synchronization, cannot be strictly defined in advance. Ultimately, user interactions and their effect on the collaborative process (that is, the syntax and semantics of the collaboration) are related to both a medium's correctness and that of other collaborative actions. As a result, a collaborative environment can be seen as an adaptive real-time system which must tailor its behaviour over the duration and application of the collaborative process.

Storage. Within a presentational system, storage of pre-recorded media data is an issue in how it affects data retrieval; how the data is delivered for processing and the characteristics of the delivery mechanism play an important part in system design and performance. This involves dealing with network issues and the need to buffer incoming data so that data is available when required to avoid degrading system performance and end-user playback. For live media streams, storage is not an issue since the data is not stored on disk but used as generated "on-the-fly" from a capture device, such as a video camera or microphone. Therefore, the flexibility afforded synthetic streams via manipulating their storage and retrieval are not possible with live data. Those systems which deal with both synthetic and live media will have to consider this difference when attempting to relate media of different modes.

Quality of Service. Quality of service (QoS) is the notion of offering a specifiable level of performance (i.e. quality) in the provision of a specific facility or service. Within the context of communications, quality of service is intended to provide guaranteed performance levels for network connectivity. A network connection, for example, would be created using a specific set of QoS parameters and that connection would then guarantee a level of performance (i.e. data delivery) based on those parameters throughout its lifetime. QoS parameters could therefore be set to match the characteristics of a certain type of network traffic and consequently function as a method to customize network connections for specific types of data (Table 2.1 [LCHS92]).

Expression of network QoS parameters is typically done using a variety of measures that correspond to the network's ability to transmit data. Typical parameters can include bandwidth, end-to-end delay, delay variation and error rate. However, while network QoS is important to achieving satisfactory media playback (by delivering media data in a manner reflective of their playback characteristics), its role within a multimedia architecture is that of an enabling infrastructure.

Medium	Typical Size/Rate	Temporal Nature	Notable QoS Feature
Text	12K/page	Discrete	Susceptible to errors
Bitmap Image	384K/Image [768 x 512 x 8]	Discrete	Susceptible to errors
Voice	8Kb/s	Continuous	Delay variation causes clicking/stuttered playback
High-Quality Audio	88Kb/s	Continuous	
Digitized Video	30Mb/s	Continuous	High bandwidth

Table 2.1: Typical QoS Parameters

Beyond the application of QoS to network considerations, there is the notion of *end-to-end QoS* which deals with the quality of how information is handled from its originating source to its actual end-user rendering. The most common example is end-to-end delay. The principle is to ensure quality throughout every aspect of the data's "life cycle". The implication is that not only the data's delivery must be managed properly but each aspect of its manipulation must be appropriately addressed. Therefore, the notion of QoS can be considered in terms of monitoring, adapting and ensuring that the characteristics and behaviours of media are correctly maintained throughout their lifetime. This end-to-end "correctness" (as applied to RASCAL) will be explored further within subsequent chapters.

In terms of system structure, the provision of QoS as an appropriately separate yet connected aspect is shown by the Lancaster architecture [LBC+92] and the QoS Broker [NS95]. Shown in Figure 2.4, the Lancaster quality of service framework forms a separate part of the architecture that interacts with other components, such as the synchronization (orchestration) service. Since media synchronization depends on data availability (and hence the importance of QoS to multimedia systems), the orchestration service needs access to the QoS framework to specify (and react to changes in) quality of service parameters. Similarly, the QoS Broker defined by Nahrstedt and Smith [NS95] provides a separate but controllable abstraction to manage and negotiate acceptable QoS levels on behalf of applications. In each case, there is an inherent *separation of concerns* in which the related aspects of the system are provided and managed separately. A similar approach is taken in RASCAL in which facets of the system and entities within it are related using a reflective meta-level methodology. Again, this notion will be further explored in upcoming chapters.

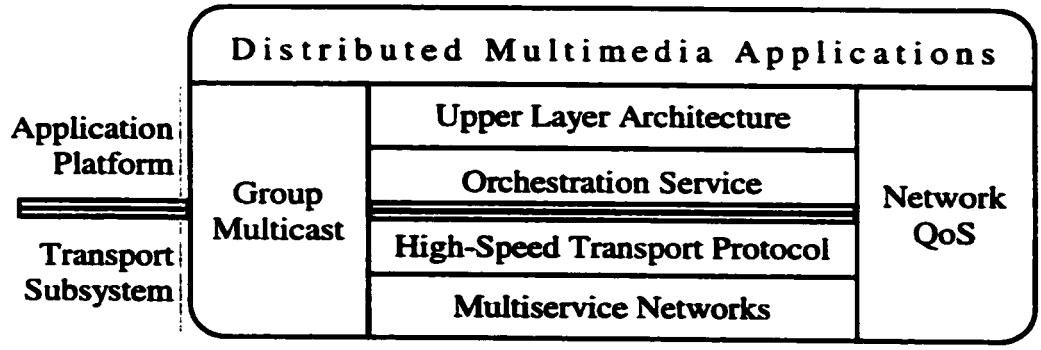


Figure 2.4: Lancaster Architecture

2.1.1.5 Collaborative Multimedia: A Motivating Synergy

The utility of employing different media is the flexibility and meaningful representation they offer to the subject at hand. By supporting varied and appropriate communication styles, they provide a naturally expressive means of explanation and way to promote understanding. Within a collaborative context, multimedia offers a valuable mechanism by which collaborators can express and exchange ideas towards realizing their goal. In short, multimedia offers a means by which collaboration can be achieved. To meet the potential needs of the collaborative process, the notion of *collaborative multimedia* combines the aspects of presentational, conversational and interactive multimedia into a single, unified and integrated framework (Figure 2.5).

Collaborative multimedia can be seen as the integration of the diverse informational formats that constitute multimedia combined with the use of computer technology to provide a shared work environment across distributed groups of users. The result is a shared collaborative media space in which participants can interact in varied and meaningful ways despite differences in locality. By employing a diverse media palette, the natural communicative styles required within the collaborative process can be supported in an integrated manner that is useful, flexible, comprehensible and meaningful to those involved.

In a traditional conversational system, interactivity is provided by using a separate teleconference in parallel with a task-specific distributed application, such as a shared drawing program. As shown in Figure 2.6, this combination illustrates the basic notion of collaboration. However, the key disadvantage of such an arrangement is the user-centric setup and coordination required in order to collaborate; that is, while the two streams of communication are occurring in parallel, they are not integrated. More often than not, such a pairing is used primarily out of necessity to ensure productive use of (i.e. "figure out") the shared application. While complementary, the use of individual, task-specific applications with no integration between them does not fulfill the potential of integrating with media within a single usage context. Rather than use a set of parallel yet

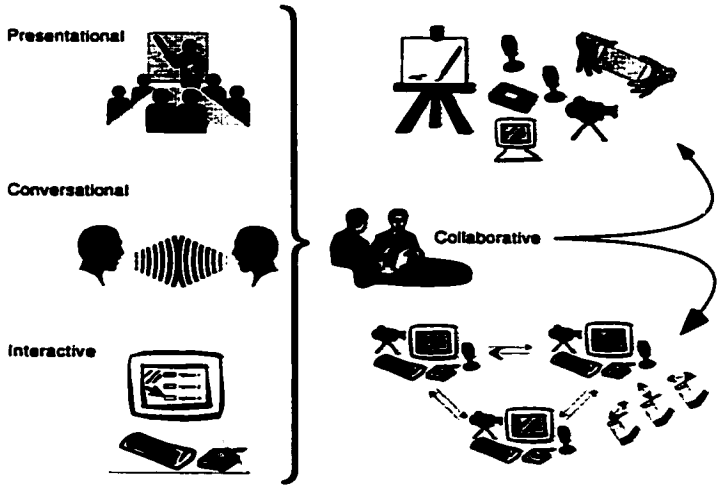


Figure 2.5: Collaborative Multimedia

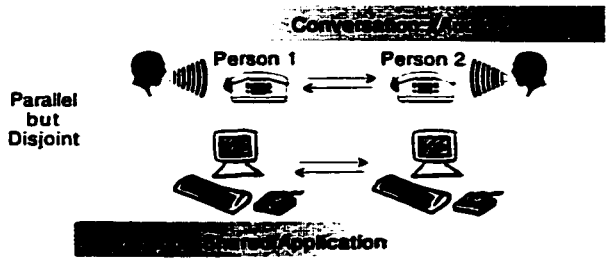


Figure 2.6: Parallel Conversation and Interaction

disjoint interactions, a collaborative multimedia system should allow participants to freely move between the types of expression, information and interaction as desired. Through an integrated environment, the system can more easily support and maintain a global perspective of participant relationships relative to the collaborative effort. That is, the system can assist the user in a more comprehensive and coordinated manner than simply offering a disjoint set of tools to use.

Ultimately, collaborative multimedia goes beyond multimedia being the *result* of a process to being *part* of the process itself: media are no longer the sole focus of the system but a means to facilitate group interaction and the collaborative exchange of ideas. Consequently, systems that facilitate such a multi-purpose environment must address both the needs of the process itself as well as its constituent technologies. For example, the media space must not only support synthetic and real-time media but also their interactive manipulation as shared objects. Consequently, synchronizing media and the actions upon them in both a timely and causal manner are necessary as part of ensuring the integrity and semantics of the individual media within the process as well as maintaining a semantically coherent workspace vis-à-vis the collaborative process itself. As will become obvious in subsequent chapters, providing for behavioural management (both real-time or not) and user interactivity forms a significant part of this work.

Table 2.2 summarizes the various aspects of multimedia in relation to the type of system under consideration. As easily seen by combining the capabilities of the mainstream multimedia classifications, collaborative systems offer the most general capabilities while also having the broadest requirements. Hence any software system that is designed to enable collaboration must be capable of providing for this diversity.

2.1.1.6 Example Systems

Over the years, much work has been done with respect to both multimedia and collaborative systems. In the multimedia domain, presentation systems like [LLBG96] and interactive systems like [SKD96] provided for rendering synthetic presentations, while conferencing systems offered enhanced telephony-like services [GA86] [Roo88]. Likewise, CSCW systems have been designed to facilitate specific types of group work; that is, specific systems supported group interaction within restricted domains. Examples include joint text editing [LHG93], electronic classrooms [SCJ87] and group decision support [KK88].

Aspect	Presentation	Conversational	Interactive	Collaborative
Data Representation	Size	Speed	Size	Both/Either
Authoring	A priori	None	A priori with Control Points	Both/Either
Storage	Required	None	Required	Both/Either
QoS	Required	Required	Required	Required
Media Mode	Synthetic	Live	Synthetic	Both/Either
Discrete & Continuous Media Support	Discrete & Continuous	Continuous	Discrete & Continuous	Both/Either
Synchronization Specification	A priori	Inherently Real-Time	A priori	Both/Either
Supports User Interaction	No	Inherent	Inherent	Both/Either
Real-Time	No	Inherent	Control Points	Both/Either

Table 2.2: Summarizing Multimedia System Considerations

In those systems that provided media-based collaboration, the projects often focused on specific aspects ranging from communication, human-computer interaction and/or psychological issues. For example, the collaborative desktop conferencing paradigm includes projects such as Rapport [AEH88], MMConf [WSM+90] and MBONE [Eri94]. Each of these support multimedia within the bounds of a specific application(s) and its (their) conversational user interface. In the case of MBONE, emphasis was also placed on network connectivity issues (in particular, its use of multicast technology). A slightly different approach to desktop conferencing is that offered by shared three-dimensional environments, such as GreenSpace [MFK+95] and SPLINE [WAB+96]. These systems promote interaction between collaborators using an *avatar*-based paradigm in which the users can combine conversation with a shared synthetic graphical environment (often called a "virtual world" or "virtual environment"). A related approach offered by multimodal environments (MEs) [CF99] blends support for synthetic and live media in a virtual environment. Other efforts provide a toolkit-like approach [GR98] [Dou96] which address collaboration from a (low-level) constructionist viewpoint. That is, rather than provide the actual collaborative environment, these projects provide *application programming interfaces* (APIs) from which to build them.

2.1.1.7 Problems and Open Issues

While the above systems are interesting and significant in their own right, they highlight a number of problems with existing approaches. In general, the main difficulties are: rigidity, integrability, dynamism and onus of responsibility.

Rigidity. In many cases, CSCW systems have limited functionality or are tightly coupled to specific types of tasks [EGR91]. Part of this difficulty is that many systems and/or applications make specific assumptions about the hardware, software or human-computer interface used with them. They also pre-suppose restrictive usage logic (i.e. behavioural rules and operation contexts) and do not provide a way to integrate themselves into a

larger whole, be it part of a larger software system or larger usage context. They provide a narrow, rigid view of the world and their role within it.

Integrability. As a result of their rigid structure, it is difficult to combine many tools not specifically designed to do so. This lack of functional integration is in part due to technical issues such as user interface management but more important is the lack of semantic compatibility. That is, the tools' behaviours are designed to be used solely in a certain way. An example is whether or not an editor supports simultaneous editing by multiple users or is strictly a single-user application. This operative context is an implicit part of the application's behavioural definition (which is almost always directly embedded in its implementation logic). How to address the provision of malleable behaviours for applications across different usage contexts and integrate them together is an important issue in being able to use "the right tool for the right job" within a collaborative context.

Dynamism. A long-standing problem with the design and provision of computer systems is support for dynamism. In real-life, it is quite common to add and remove entities from their surroundings when necessary; and even more common is the ability to "change your mind" and do things differently depending on context and how a situation evolves over time. Doing so in software, however, is more problematic. While the dynamic structure of software systems has received increased attention in the last many years with the advent of component-based software ranging from COM objects [COM95] to Java applets [Sun00] [Fla99], the ability to provide dynamic behavioural modification within the context of fixed structure is still difficult. In particular, behaviours are typically either fixed or chosen from a preset list. Additionally, rather than these behaviours being switchable "on-the-fly," they are often statically maintained for the duration of an entity, effectively requiring an object to be recreated with the new behaviour (i.e. restart the application, reboot the machine, etc.).

Onus of Responsibility. Within the majority of collaborative and multimedia systems, support for aspects such as dynamism and integrability is not ubiquitous. And often when provided, the responsibility to enforce and manage them is that of the user. An example is the MBONE suite of applications, which can provide a collaborative environment like that in Figure 2.6. While each application provides the appropriate base functionality, their integration into the collaborative process is not supported within the MBONE toolset; it is strictly up to the user to enforce any "semantic connection" between tools being used at the same time. Doing so, however, can be a cumbersome burden on the user which gets magnified as the collaborative context becomes more involved. In particular, it decreases the seamless fluidity that is a necessary part of successful collaboration and forces the users to constantly perform "semantic shifts" between the collaborative task and management of the collaborative applications themselves. Such a requirement can therefore become a limiting factor affecting the degree to which a user will utilize a given tool set. Ultimately, it is a reflection of a system that is not able to abstract itself to the appropriate level in which users can exercise control in a detached "semantic declarative" fashion rather than in an obtrusive "manipulative procedural" manner.

In summary, a collaborative multimedia system must attempt to address these issues as an inherent part of its definition. As will be shown in subsequent chapters, the RASCAL framework provides for ways to address these issues in a more complete and orthogonal sense than many of the existing approaches.

2.1.2 Behavioural Management: Coordination and Synchronization

As a phrase perhaps more at home in psychological circles and human resources sectors, "behavioural management" refers to the controlled organization of activities that some entity performs, both "in the large" and "in the small". In terms of computer systems, this means that activities are organized across levels ranging from high-level semantics to low-level mechanisms. And as the activities that people perform constitute their behaviour, so too does the functionalities of a computer system constitute its behaviour. Therefore, the controlled organization of that functionality maps to the coordination and synchronization of activities that make up that functional behaviour.

2.1.2.1 Synchronization

As the most familiar of the behavioural management concepts, synchronization deals with the lowest levels of behavioural management. By constituting management in the "small," synchronization is the "micro management" of specific fine-granular details. A general overview of synchronization, along with multimedia specifics now follows.

2.1.2.1.1 General Principles and Their Application to Multimedia

The general concept of synchronization is applicable across a wide range of natural, technical and social systems. Typically, it is characterized by relating the occurrence of some activity (i.e. event) to that of another. In particular, synchronization can be considered both in terms of time (i.e. temporal) as well as event order (i.e. causal).

While any general discussion of synchronization should address a number of significant issues, traditional multimedia synchronization typically only referred to the maintenance of temporal relationships between media. To meet the needs of collaborative multimedia, however, these aspects become increasingly important. Consequently, timeliness, precision, causality, granularity and paradigm, in addition to the role of awareness and media mode within multimedia systems, are hilighted. The reader is referred to Appendix A for an overview of the nomenclature used with respect to media organization and multimedia-specific synchronization.

Timeliness and Precision. Traditionally, the notion of synchronization has referred to the alignment of event occurrence with a specific point in time; that is, an event was considered synchronized if it occurred in a timely manner. Such a notion of synchrony is that exemplified by an alarm clock or metronome and is more precisely described as *temporal synchronization*. In relation to timeliness, precision refers to the accuracy at which event occurrence must be aligned to the given point in time; that is, how close an event must occur relative to the correct time. Therefore, the notion of precision relates to the maximum allowable error by which an event can be considered in synchronization. This notion is also known as *tolerable asynchrony*.

As applied to multimedia, the notion of timeliness is applicable to many levels, such as network QoS guarantees for receiving data and the availability of media units (LDUs) at display devices. Variations from the appropriate time results in asynchrony and can lower system quality as perceived by the user. The compounded effect of asynchrony also needs to be minimized so that synchronization failure at specific points in the object's manipulation do not "ripple" through subsequent processing stages; that is, there needs to be sufficient isolation between stages so that asynchrony at one level is not directly passed on to another. In terms of precision, however, pin-point accuracy is not always required in order to preserve semantics; rather, a tolerable degree of asynchrony at the user interface is acceptable. For example, audio/video tracks in the traditional "lip sync" scenario must render their corresponding LDUs within 80*ms* in order to appear synchronized [Ste96]; otherwise, user comprehension is degraded. Timeliness is also necessary to achieve natural and fluid communication between participants in a conversational and interactive environment.

Causality. Causality is the concept in which the occurrence of events is related by a temporal-dependency relationship; that is, if an event A is caused by event B, then B must occur before A [Lam78]. Even given the notion of timeliness, the occurrence of events "close" to each other in time is not the same as ensuring that events occur in the correct order. For example, should the chair be pulled out from underneath someone sitting down, noticing whether or not the chair was removed *strictly before* sitting down will causally affect the outcome. Causality therefore defines the acceptable and semantically valid ordering of events. Also known as *causal synchronization*, causality plays an important role in many types of systems ranging from distributed databases [CP92] to parallel discrete event simulation systems [Fuj90] [Jef85].

Furthermore, by defining the acceptable and semantically valid order of events through the use of a temporal-dependency relationship, causality plays a subtle yet pervasive role in collaboration. In a physically shared (i.e. co-located) environment, event ordering is obvious due to the mutually simultaneous observation of and participation in the shared event space. In a distributed, logically shared environment, however, such a dependency is not obvious and its realization must be ensured by a mutually acceptable technique. In a shared text editor, for example, if one user tries to delete a paragraph while another attempts to edit it, divergent views of the document develop; the result in a non-coherent environment in which participant behaviours and the state of the shared space are ambiguous. Consequently, the order of events and the effects of timeliness on the maintenance of causality have major implications with respect to the semantic coherence of interactions between participants and their manipulation of media within a multimedia system.

Granularity. In most cases, synchronization can be seen as the binding of an event's occurrence to a specific point in time, or of multiple events to a mutual point in time. This "point" may refer to a particular instance or a bounded interval which defines the *granularity* of the synchronization. The closer in time two events must occur to be considered synchronized, the finer the synchronization granularity; that is, the finer the granularity, the tighter the bounds on the *divergence* allowed between the events. The interpretation of divergence is dependent upon the system in question; within a conferencing systems, the divergence would refer to mismatched audio and video tracks (i.e. "lip sync") while within a replicate database environment, the divergence could refer to replicas having different values (value divergence) or how long it takes to update all

the replicas (temporal divergence) [CP92]. Depending on the application area, various degrees of divergence are acceptable and in part define which approaches to synchronization are appropriate in a given situation.

In terms of multimedia, if a given media object is taken as a stream of (LDU) display events, a continuous medium forms a series of periodic events while a discrete medium constitutes a single sporadic event. In this sense, media synchronization can be viewed as the coordination of events, in both a timely and causal manner. It then follows that these events could be generated by different sources, such as media stream, a management system or a user interface. In cases where events originate from a single source, event ordering naturally follows event generation. However, in a system in which interacting events originate from multiple sources in a (possibly) interleaved fashion, maintaining causality becomes more difficult. Therefore, the frequency at which events can occur can impact the degree to which the system can efficiently address their synchronization.

Paradigm. One of the least familiar aspects of synchronization is that of paradigm, which refers to the "manner" in which synchronization is enforced. There are two broad classifications for synchronization paradigm: pessimistic and optimistic [EN94] [Jef85]. The *pessimistic* (or *conservative*) approach is based on the assumption that numerous temporal (ordering) conflicts between events are likely; consequently, in such situations it would better to postpone any actions that are likely to conflict until such a time when conflicts are not possible. That is, the pessimistic approach assumes the worst possible interaction between events and is therefore "cautious" about their execution to avoid prematurely performing actions out-of-order. The *optimistic* approach, however, is based on the assumption that such temporal (ordering) conflicts are rare and that the overhead and inefficiencies brought about by trying to avoid them are worse than simply executing (and later undoing) the activities in the first place. Those situations that result in a temporal disorder and end up "out of sync" are subsequently "undone" so that their effects on the system state are erased.

Pessimistic techniques can therefore be seen to delay the occurrence of events while optimistic mechanisms allow processes to execute forward at their own rate [KR81]. As shown in Figure 2.7, conflict detection is performed early on within the pessimistic approach, prior to starting the "read/compute/write" cycle; in the optimistic scheme, however, conflict detection can occur at any time up until immediately before execution of an activity whose effects cannot be reversed, such as a "write".

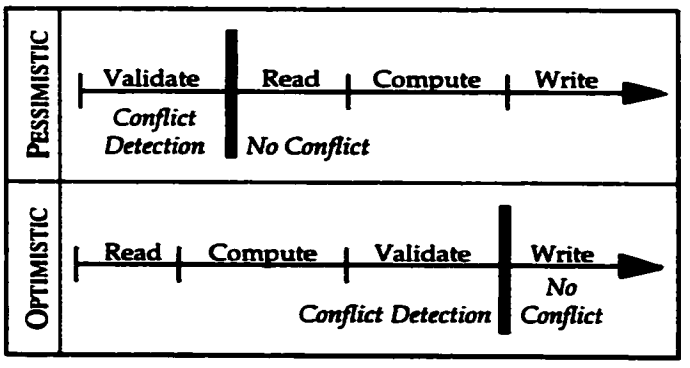


Figure 2.7: Pessimistic/Optimistic Conflict Detection

These two approaches are mirrored by two classes of synchronization algorithms: those based on *locking* and those based on *protocols* [EN94]. Algorithms that use locking are based on the notion of mutually exclusive access to some shared entity and are pessimistic in nature. Since locking involves the exclusive control of a resource for a period of time, all other attempts to control that resource are denied until the lock is released; hence, locking defers possible conflicts by disallowing concurrent access. Protocols, on the other hand, are based on the notion of rules that order execution. Consequently, protocols could be viewed as more (but not exclusively) optimistic in nature; effectively, they define a public interface for the manipulation of a resource and the regulations by which conflicting interactions are resolved.

Pessimistic techniques [BG81] [BHG87] are essentially one of three kinds: lock-based, timestamp-based or a hybrid of these mechanisms. The advantage of such techniques is that they are very common and well-understood. The notion of locking is well-suited to everyday interactions and provides a well-known paradigm for controlling access to shared resources. It is a well-utilized technology common to operating system and centralized database technology. However, its disadvantages are magnified when used in a distributed system. First, the overhead of lock management (creation, deletion, setting, releasing and so forth) can be significant in general and more so when it needs to be done in a networked (i.e. distributed) manner. Similarly, with the use of locks, the possibility of deadlock is always present; there is no locking technique which can be completely deadlock-free. Therefore, in a distributed system, both local and distributed deadlock must be taken into account. Addressing these issues can result in significant overhead both in terms of systems complexity and performance. Of particular concern is the strictness imposed by the use of locking. In order for locking to be an effective technique, all resources must be subject to locking at any point a conflict may occur. Consequently, locking tends to be a frequent activity with a wide range of granularity (i.e. done on a wide variety of differently sized objects) irrespective of whether locking is actually needed or not. Consequently, in a concurrent system, locking restricts and lowers potential concurrency because even those resources that do not necessarily need protection for a given activity are usually locked anyway. Those systems based on timestamps likewise forbid actions whose timestamps may conflict from being executed until no such conflict is possible. In such systems, potential concurrency is also restricted and the overhead to analyze the possibility of conflicting actions can be substantial.

Optimistic techniques [KR81] [Här84] [Jef85] are based on the notion of *histories*, timestamps and/or some combination of these methods. The advantages of these techniques are that they utilize any potential concurrency available within the application, avoid the overhead of lock management (since no locks are used) and are consequently deadlock free. Such characteristics lend favourably to the use of optimistic techniques within a distributed environment. Because optimistic systems tend to incur most system management overhead during *rollback* (i.e. the "undoing" of computations), the costs associated with distribution of this management are therefore consolidated as opposed to being broadly accrued over time. Furthermore, since the underlying assumption of optimistic systems is the lack of rollback-inducing conflicts, such management overhead (e.g. network messaging and so forth) is yet further reduced. Unfortunately, the expense of actually performing

rollback and maintaining the required history mechanism can be quite high. Also of note is that depending on the algorithm used, issues of livelock and starvation must be taken into account.

While the preceding factors may influence the selection of a synchronization mechanism, the effect of synchronization paradigm on the user interface is also an important issue to consider in multimedia system design [GM94]. Systems in which synchronization errors are regarded as severe are arguably best by a conservative (or pessimistic) synchronization policy which does not allow events to occur until no possibility that an error in ordering would occur. In a volatile and unpredictable environment, however, such knowledge is unlikely to be available. Consequently, an optimistic synchronization mechanism that allows for out-of-order events and subsequently corrects them as necessary is preferred. Since optimistic synchronization is based on the premise that few synchronization errors occur, it tends to more naturally fit human-centered designs [GM94] by complementing the use of social conventions which prevent many synchronization anomalies before they occur. In a technology-centered design, however, synchronization errors must be avoided strictly by algorithmic means; therefore, the application of a conservative mechanism to avoid the introduction of event disorder is required. Consequently, in a system which integrates both of these worlds, support for both paradigms is more suitable. A serious difficulty is, however, how to support a valid user interface(s) in a system that supports differing paradigms.

Awareness. In non-shared environments, like those typified by presentational multimedia systems, individual users only have to deal with their own actions. Within a shared environment, such as a collaborative system, participants need to be aware of how their behaviours "fit" with that of other users. Therefore, the issue of awareness becomes fundamentally important in facilitating real-time interactive exchanges in non-physically shared environments [GG95]. In a physically shared space, collaborators can physically observe, encompass and react to the entire space at a single instant. All participants can simultaneously evaluate their common environment and adapt to their circumstances; that is, the participants are universally aware of shared space state. In a non-physically shared environment, however, collaborators do not implicitly have this simultaneous awareness; therefore, it must be provided. This necessary acuity of the shared environment is an important element in which all participants can relate to activities at the same time. In order to interact in an equitable manner, activities must occur in a synchronous manner across all participants [AY96]. That is, users cannot properly relate to other behaviours and actions if they are not aware of them. Support for awareness must be provided in a manner that enables users to appropriately synchronize themselves with other entities in the environment.

Media Mode. The applicability of synchronization techniques is often based on media mode. For example, presentational systems that deal solely with synthetic media may employ mechanisms that are not suitable for conversational systems which provide real-time, live media support. While playback behaviour within presentational systems is known *a priori*, this is not the case where media are generated "on-the-fly". Preventative synchronization techniques (such as buffering and decoding media in advance of playback) are therefore not applicable in a conversational setting like they are in presentational systems. Consequently, media mode becomes a limiting factor in the use of corrective synchronization mechanisms as part of ensuring fluid

interaction. Additionally, systems which combine media of different modalities must support and be able to compensate for such differences and their effect on processing and fluidity within the system.

2.1.2.1.1.1 An Illustrative Example

The interaction and mutual importance of the above facets can best be illustrated using a real-world musical scenario (which itself will be illustrated as an example media space in subsequent chapters). Within a musical performance, the correct rendition of musical elements (i.e. notes/rests) relates to the synchronization of events. Causality refers to the ordering of the notes and rests while timeliness and precision refer to their correct duration. The grouping of the musical elements together in time (such as triplets, polyphonic intervals and chords, etc.) relate to the notion of granularity as to how the elements are "played together". Additionally, when musicians are playing in groups, awareness of each other is fundamental to achieving a cohesive and artistic performance. While the accuracy achieved by following the musical score can be seen as employing pessimistic synchronization, the "counting on" other musicians to perform accordingly and subsequently correcting/allowing for mistakes and variations can be seen as applying the optimistic paradigm.

2.1.2.1.2 Multimedia Synchronization: Basic System Concerns

Two of the most beneficial aspects of a multimedia system, its flexibility and diversity, can also be seen as two of its most difficult. In order to convey the meaning of a particular medium, both its individual temporal requirements and those with which it is combined must be considered.

For example, a presentation may specify the concurrent display of a text string with a video clip for n seconds, followed by an image. Doing so requires specifying the temporal relationships between the media objects as well as deriving their playback schedule. To ensure that playback conforms to expectations, a synchronization mechanism must be used to enforce it. For example, large-grain synchrony at an object level must be provided in order to correctly begin playback. Media objects within a presentation, however, may have different temporal dimensions (i.e. continuous vs. discrete). Therefore, the synchronization mechanism must support multiple synchronization granularities so that continuous media can also be rendered correctly.

Consequently, in parallel with the need to specify synchronization requirements, there is also the need to provide for mechanisms to enforce them. In this regard, there are two primary system-level questions that must be answered: (1) what kind of mechanism(s) is (are) required; and (2) where should the mechanism(s) be placed.

2.1.2.1.2.1 Kind of Mechanism

Two general types of synchronization mechanisms were suggested by [LT91]: (1) synchronization markers; and (2) synchronization channels. ***Synchronization markers*** function as tags by which media streams can correlate their temporal position during rendering. These tags effectively mark off sections of the media stream and can be transmitted as part of a raw data stream or generated externally and "imposed" on the data stream.

For example, a video clip could be transmitted as individual frames with inter-frame markers inserted between each frame; the SMPTE (Society of Motion Picture and Television Engineers) code used by high-end video equipment is an example of synchronization markers that are embedded within each video frame itself. The use of *synchronization channels* is designed to isolate the control achieved via synchronization markers embedded within the media stream to another communications channel running in parallel. The control information within the synchronization channel contains references to the data transmitted in data-only channels and directs the synchronization mechanism at the receiver as how to align the data.

Numerous specific approaches [Gib91] [GDT91] [AH91] [BCC+92] [CCGH92] [LK92] [RR92] [LKG93a] [LKG93b] [WQG94] [Rob95] to media synchronization within presentational systems can be found in the literature. In general, the underlying mechanisms can be characterized as follows:

- *layered construction*: Synchronization is addressed at multiple stages, using different entities and mechanisms to correct asynchrony as it becomes noticeable (e.g. the network and playback levels, within and/or between streams, etc.).
- *object abstraction*: Media data and system components are modeled as independent but interacting objects.
- *event synchronization enabled through scheduling*: Coarse-grain event-level synchronization is facilitated by using scheduled media object playback times.
- *continuous synchronization enabled through fine-grain temporal intervals*: Fine-grain continuous synchronization is achieved through the division of a media object into a series of small temporal sub-divisions which are individually aligned to their correct playback time.

Conversational systems, on the other hand, tend to rely more on protocol-based techniques. Examples include the MBONE [Eri94] audio tools *ivs* and *vat* which use the RTP [SCFJ96] protocol. In these systems, the synchronization mechanism functions as a protocol engine ensuring that the data conforms to the protocol. This is in contrast to presentational systems in which the mechanism ensures the data conforms to an external specification (i.e. the presentation's scenario timeline).

2.1.2.1.2.2 Location of Mechanism

In a distributed multimedia system, there are three possible locations for the synchronization mechanism to reside: the media source, the receiving site and the network connecting them.

At the Media Source. Synchronization at the media source implies that data is synchronized before transmission to the receiving site. Consequently, the temporal relationships imposed prior to transmission must be maintained during transmission and up until playback at the receiving site. A fundamental assumption of this method is that all the media to be synchronized are located at the same source and can be "wrapped up" or multiplexed into a single data stream. However, these assumptions may not always be realistic, desirable or

amenable to the application of the system. As well, this technique implies an intelligent source that is knowledgeable about the media it is providing and well as its intended use at the receiver.

Within the Network. Synchronization within the network has traditionally been viewed as a protocol-based approach that formed an elaborate "hand-shaking" mechanism with precisely defined timing [LT91]. As such, it proved a complex technique that was network-dependent and prohibitively difficult when a large number of media streams became involved. As with source-based synchronization, this technique also assumed an intelligent source and a receiving site that will do nothing that could somehow induce asynchrony between the streams. For these reasons, this approach has traditionally been unpopular.

At the Receiving Site. Synchronization at the receiving site (sink) enforces the required temporal constraints on each media stream after it is received from its source over the network. This kind of synchronization mechanism allows each media stream to originate from its own source and does not require any inter-stream management by the network. However, it can place a large processing burden on the receiver and provides (as with source-based synchronization) a solitary approach to the synchronization (and coordination) of media and their interactions. However, because of difficulties associated with transport induced delays, locating the synchronization mechanisms at the receiving site as traditionally been the most popular technique.

While generally useful, the difficulty with the above categorizations is the implicit notion that they must be used in isolation rather than in a complementary fashion. For example, by combining network and receiving site synchronization, a multimedia system could reduce the complexities of the receiver's synchronization mechanism while still allowing for processing not normally supported by the network model.

The major problem with this taxonomy, however, is that it assumes the traditional computer vs. network model: the source and sink are the computational entities while the network is a "dumb" transport agent. However, when the notion of *intelligent networks* [ITU92] and/or *programmable networks* [CKV+99] is applied, network synchronization can be transformed into a *network service* based on a computationally-enabled "smart" network rather than simply constituting a network protocol. Such an idea is the basis of the work presented in this dissertation as is presented through the use of a coordinated communications subsystem and the collaborative multimedia support system (CCS/CMSS) as discussed in upcoming chapters.

2.1.2.2 Coordination

In addition to synchronization, a system must also deal with higher levels of behavioural management. By constituting management in the "large," coordination is the "macro management" of higher level entities, their interactions and the semantics governing their use. To explain this notion, we first offer a general overview of coordination, followed by a discussion specific to collaborative multimedia systems.

2.1.2.2.1 The Role of Coordination

As described in [MC94], the coordination problem can be summarized as the *"integration and harmonious adjustment of individual work efforts towards the accomplishment of a larger goal"* [Sin89]. However, the importance of coordination is often overlooked because it is such a prevalent, subtle and assumed facet of everyday existence. Indeed, the notion of a coordinated system spans many disciplines and can be applied at many different levels within them. However, regardless of its specific application, coordination highlights the need to maintain a coherent system state in which participating entities are aware of where, when and how they fit within the "grand scheme of things". While often a presumed axiom of human-centered activity, coordination in a technological environment requires more pedantry. In mixed human and technological systems like that of a media space, coordination needs to function in a supportive and adaptive role. The specifics of coordinating an environment need to depend on which entities are involved and the semantics of its usage scenario. Such a situation-dependent approach is very common in daily life as "the rules of the game" are constantly and dynamically adjusted to the situation at hand. Therefore, coordination facilities that support such an approach can be said to offer *context-mediated behaviours*.

As part of achieving a coordinated environment, synchronization plays a fundamental role. Defined as "the representation or arrangement (of events) to indicate coincidence or coexistence" [Web89b], synchronization can be seen as a necessary part of "making sense" out of a series of actions. However, as synchronization tends to examine the actions in terms of component entities, it does not (via its level of approach) take into account the more abstract meaning of interaction of an entity as whole, as interacting composites, in relation to some goal or in any particular semantic context. For example, consider an audio/video voiceover combined with a text caption and accompanying visual. Even if the pair of audio/video streams are perfectly synchronized, if their content does not match the appropriate text or is shown with incorrect imagery (i.e. is not coordinated relative to the other elements), the presentation as a whole will not make sense. In other words, there will be a *semantic discontinuity* and the intended meaning of the presentation will be lost (see Figure 2.8). Synchronization can therefore be seen as a necessary but insufficient level of behavioural management within systems that have a high degree of semantic complexity – a collaborative media space being one such example. Shown in Figure 2.9, collaboration, coordination and synchronization are mutually dependent upon each other

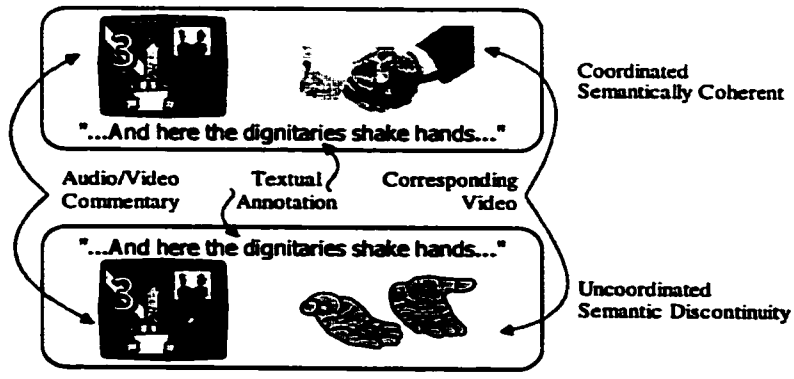


Figure 2.8: Coordination and Semantic Coherence

in terms of abstracting behavioural complexity and offering a way to relate the characteristics of behavioural management to different levels of abstraction. In essence, coordination can be seen to function at a *meta*-level relative to synchronization, whose separate consideration offers increased semantic, conceptual and implementational clarity.

2.1.2.2.2 The Need for Coordination in Collaborative Multimedia

In so much as collaboration is itself an abstract concept, so too are the benefits of collaborative multimedia realized at an abstract level. The need to combine synthetic and live media of both conversational and interactive varieties, for example, introduces the need to address mixed mode synchrony and the dynamic formation of interacting media object groups. Further complexity results from the need to provide for multi-synchronous groups, which are semantic "associations" marked by disjoint synchronization needs. For example, associating real-time audio commentary, synthetic video clip playback and interactive text document editing. By addressing the behavioural management of such complex collaborative scenarios at strictly a "synchronization level", low-level solutions to high-level problems results in a complex tangle of low-level techniques amplified by still insufficient means to deal with the higher level semantics.

These kinds of issues, including related scenarios and example constructs will be illustrated further in subsequent chapters.

Based on existing systems, the problem shared by most approaches to media synchronization and shared environment coordination is two-fold: (1) excessive protocol-orientation; and (2) rigidity in the area of application. The majority of traditional approaches to multimedia synchronization were based on the use of protocols and/or a timeline/scheduling mechanism. Typically, protocols were used in distributed real-time applications (such as conversational multimedia) while scheduling was used in presentational applications. However, the problem with protocol approaches are that they do not scale well (i.e. they become excessively complicated) when the number of entities gets large and/or the synchronization parameters (such as different granularities, paradigms) vary. In fact, a single protocol to handle the potential heterogeneity of synchronization requirements within a collaborative multimedia environment, given the varied characteristics would be prohibitively complex, if at all possible. While the use of transport or application level protocols

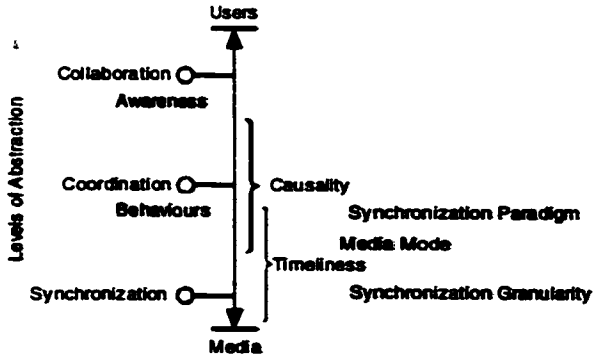


Figure 2.9: Aspects of Collaborative Behaviours

(such as RTP, etc.) are necessary, they are not a sufficient or necessarily efficient way to build *arbitrarily* complex systems. Similarly, the use of scheduling mechanisms is not conducive to the real-time data generation typical of collaborative interaction and conversation. An attempt to combine the two approaches in one (effectively nesting the two techniques to achieve a "protocol schedule" or "schedule protocol") would at best be unclear and cumbersome.

Consequently, there is need to utilize the various "tried and true" techniques where they fit and where they work, rather than attempt to use inappropriate ones in ways and at levels for which they are not suited. Hence, this dissertation can be seen to offer an integration framework that allows for this integration through the novel combination of system engineering techniques.

2.1.3 Systems Engineering

In order to meet the diverse and open-ended requirements of the preceding approaches, this section overviews a number of systems engineering techniques and principles on which this research is based. These include: real-time systems, software engineering, middleware, meta-technologies and reflection.

2.1.3.1 Real-Time Systems

A *real-time system* (RTS) is defined as one whose correctness depends not only on its logical results but also the temporal properties of its behaviour [LA90]. The basic notion of real-time systems is well-known and such systems have been used for many years. Also known as *event-driven systems*, real-time processing uses the *event-response* paradigm while traditional data processing considers output as a logical function of its input (see Figure 2.10). Examples include the control of factory robots as well as avionic subsystems in airplanes and are characterized as: deterministic, having timely responses (based on guaranteed deadlines), offering flexible scheduling abilities, having a very high demand for security, requiring low fault tolerance and being robust under all conditions [SN95]. For example, unanticipated delays in a robot welding car parts together could result in the parts being welded incorrectly as well as the robot itself being damaged. Such severe consequences therefore necessitate that events occur only at their scheduled time.

Real-time systems can be seen not necessarily as "fast" but as "temporally pedantic". That is, the behaviour of a real-time environment must occur *on time* since early event occurrence can be just as damaging as incurring delays. However, the severity associated with missing a scheduled deadline varies and systems are therefore classified as either "hard" or "soft" real-time.

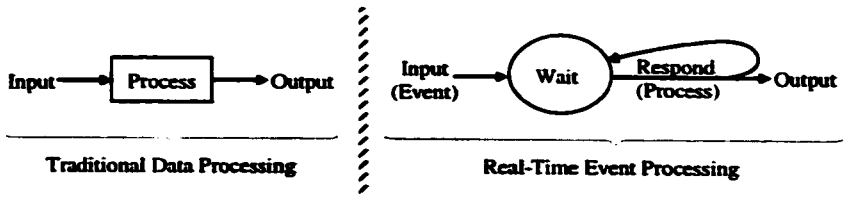


Figure 2.10: Transactional vs. Real-Time Processing

Hard real-time systems are those in which any violation of a timing constraint is considered a system failure. Consequently, timely execution must be guaranteed through resource allocation based on the worst possible situation. Such a strict policy usually results in under-utilized resources during normal operation and can require a complete system shutdown should any anomalies be detected. The robot and avionic systems discussed above are examples of this type.

Soft real-time systems, however, are those in which a violation of a temporal constraint does not constitute a system failure. In contrast to the above examples, multimedia systems generally constitute soft real-time systems in which temporal performance requirements are usually not so restrictive [SN95] [Bur94]. Typically, if an object in a presentation is displayed behind schedule, the consequences are comparatively minor: the quality of the presentation may be degraded and the user may be annoyed or confused by the asynchrony – but no physical damage results[2]. Relatedly, the human-centric character of multimedia systems implies a range of "acceptable" playback quality which varies with individual users. Consequently, human perceptual limitations can be used to relax certain timing constraints within a multimedia system [SE93]. This facilitates a choice between which playback characteristics are most important and possible trade-offs between functionality and resources usage. For example, if the bandwidth of a video channel is constrained to only 15*fps* at a specific resolution, the user could decide to accept the provided quality or adjust select parameters more aptly suited to his/her needs. In addition, the isochronous nature of continuous media allows continuous media streams to be viewed as a periodic task [SN95]. The result is more flexibility in scheduling which facilitates easier resource handling, which is extremely valuable when considering the resource-intensive demands associated with multimedia data. Subsequently, the soft real-time nature of multimedia illustrates the need for an adaptive approach to system management.

As part of such an approach, three important aspects must be considered: system support, structure and monitoring. *System support* for real-time environments includes providing the appropriate hardware/software infrastructure, including provision of a real-time operating system [LA90] [GA91] [BL91] [Ste95]. However, rather than provide an in-depth discussion of such issues, it is simply noted that a real-time media space architecture needs to consider its computational environment. In terms of *structure*, the software engineering practices applied to the design and implementation of a real-time system are also important [LA90] [SR95]. This is true both in terms of providing a flexible and extensible architecture capable of real-time performance but also one in which the actual performance can be monitored [Jah95]. In order to provide an adaptive system, it is necessary to have a system which can *monitor* its own behaviour during run-time. Therefore, the system can evaluate any behavioural problems and take the necessary measures to correct them. To be useful, monitoring must be done in a non-obtrusive manner and not adversely affect system operation. As applied to media space behavioural management, these issues need to be addressed in a manner which facilitates rather than hinders the collaborative effort.

---

[2] An obvious exception would be a tele-surgery application in which a lack of temporal accuracy could jeopardize a patient's health.

2.1.3.2 Software Engineering

An increasingly pervasive term, software engineering is commonly used by many to refer to the process building software. However, the proper definition of software engineering is more broadly based to refer to "...the systematic approach to the development, operation, maintenance and retirement of software" [Vli93]. Consequently, the true importance of software engineering lies in the fact that it addresses the software lifecycle in a complete sense, so as to meet the overall needs of specific systems.

As introduced in the previous section, real-time systems have very specific requirements in terms of their behaviour and in meeting the needs of their operating environment. Consequently, software engineering plays a major role within real-time systems, and in part explains its importance to this research. In particular, the work presented in this dissertation borrows from five particular areas of software engineering in terms of its philosophy; these are: patterns, frameworks, lifecycle management, object-orientation and executable modelling.

2.1.3.2.1 Patterns and Pattern-Based Architectures

As an increasingly popular element within the software engineering community, *patterns* are a concept which originated in the field of building architecture in the 1970s [Ale79] [AIS77]. Used originally in a spatial (i.e. architectural) context, the original notion of a pattern was specified by Christopher Alexander as follows:

> "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves. As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant."

In essence, a pattern can be seen as a kind of *template* used to specify how a general solution is applicable within a given context for a particular problem. In software engineering, patterns have been applied to the design of computer-related systems, ranging from *architectural patterns* which express system organization, *design patterns* for detailing subsystems, and *idioms* which hilight particular implementation constructs [GHJV94]. In effect, they are common solution strategies that appear again and again, and include well known examples such as client/server and layers (see Figure 2.11). While a thorough discussion of patterns is beyond the scope of this work, Figure 2.12 summarizes a variety of architectural and design patterns that will be referenced when discussing the proposed RASCAL framework in subsequent chapters.

The use of patterns in the realization of *pattern-oriented software architectures* [BMR+96] has extended the concepts of encapsulation and re-use from simple code modules (such as procedures) to entire subsystems of interacting components (i.e. component-based systems). Such an approach, commonly referred to as *componentware*, utilizes each component as an independently developed and tested entity with well-defined interfaces to connect to and function as part of a larger system. Often discussed in relation to frameworks

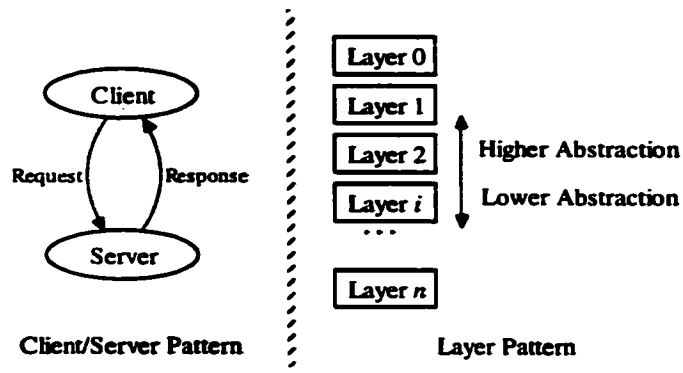


Figure 2.11: Client/Server and Layer Patterns

[FS97] [Joh97] [Sch97] [App00], this integration can be addressed in two ways: (1) structural; and (2) behavioural.

- The *structural view* of a component-based architecture is one in which an entity's functional role and interconnections are governed according to a specific well-known configuration. This configuration can be seen as a pattern which specifies the arrangement of these entities. Therefore, within a collaborative multimedia system, this view would address its design as a pattern(s) of collaborating entities, each of which support/enable a particular kind of collaborative activity or multimedia functionality. For example, the playback of a synthetic movie would typically employ the client/server pattern through the use of data server and playback client entities.
- The *behavioural view* of a component-based architecture constitutes the logical perspective of entities and their interaction as part of a particular usage scenario. Within the collaborative process, the interaction between participants and the media they use is a highly purposeful dialogue in which participants' activities and the exchanges between them are part of a mutually understood "goal-oriented" behaviour. Therefore, as mentioned in the introduction, participants can be seen to engage in specific collaborative *behavioural patterns*[3] which guide their mutual understanding and activities, both as individuals as well as members of a collaborating group. Complex behaviours are a combination of simpler ones, with each behaviour being a pattern of entities which interact in a specific well-defined manner. The result is a composition of complex patterns from simpler ones, functioning much like *behavioural components* in the overall collaborative effort.

In his original work, Alexander hierarchically structured his patterns into a system called a *pattern language*. Reminiscent of botanical vines left to find their own way, Alexander's arrangement was capable of adapting its form to suit the particular needs of a given situation, deviating from the set path when suitable. Such an

[3] This use of "behavioural" does *not* mean the same thing as in [GHJV94]. Their behavioural patterns are a subset of design patterns, in which behaviour is implicit to structure. Our use of "behavioural" is explicit and at a higher level, being orthogonal to the architectural/design/idiom classification.

adaptive nature can also be applied beyond the structural (i.e. spatial) arrangement, in terms of behavioural management. Therefore, when applied to the use of *behavioural patterns*, combined with the application of reflection and intelligent networking principles, the result is the notion of *intelligent self-investigative behaviours*. This idea will be explored further in upcoming chapters.

2.1.3.2.2 Frameworks

As an approach to software development, frameworks have been characterized as "reusable mini-architectures" that provide the basic structure and behaviour for a set of abstractions within a given domain [App00]. By supplying infrastructure and mechanisms to effect interaction between abstract components with open implementations, a framework offers architectural guidance by partitioning the design and defining responsibilities and collaborations between entities [GHJV94]

Most often, frameworks are viewed in a programming-only context; examples include MFC – Microsoft Foundation Classes [Bla97] [Tem97] and ATL – the Active Template Library [Box97]). While suitable examples, frameworks are actually a broader, more generic and powerful design technique whose importance is shown by a growing body of research [FS97] [Joh97] [Sch97] [DMNS97] [BMA97] [CHSV97]. Domain-specific examples also include those that deal with multimedia and user interfaces [Ack94b] [PLV97].

As applied to this research, RASCAL takes a framework-based approach to providing for behavioural management within collaborative multimedia systems. By addressing the system's software engineering in a reflective manner (as discussed shortly), the framework also provides a way to deal with the management of system's whole lifecycle.

2.1.3.2.3 Lifecycle Management

Dealing with the on-going reality of computer systems is an important facet of software engineering. This aspect, known as the *software lifecycle*, emphasizes all phases of software development, from requirements analysis and design, through to implementation, maintenance and retirement. And as applied to real-time systems, lifecycle management plays an important role in ensuring that not only the initial design of the system meets its real-time needs, but also that its continued evolution satisfies them as well. Consequently, a real-time system like that of a collaborative media space with highly dynamic behaviours that evolve over time must pay considerable attention to such issues.

The traditional *waterfall model* is a software development process that structures analysis, design, programming and testing as four distinct steps, each of which is completed before the next step begins. Consequently, it implicitly assumes a static system which does not change, and is therefore not suitable for a real-time system which must be adaptive to its environment. The technique known as the *spiral methodology*, however, offers an incremental improvement over the waterfall model and is more suitable for real-time systems. While still consisting of four phases, rather than complete each step strictly in turn, the spiral methodology spends an initial amount time in each phase, followed by several iterations over all four phases.

	Pattern	Usage/Comment
Architectural Patterns	Blackboard	Useful for problems for which no deterministic solution strategies are known. In this pattern, several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution.
	Broker	Used to structure distributed software systems with decoupled components that interact by remote service invocations. A broker component is responsible for coordinating communication between components.
	Metamorphosis	Used to provide run-time modifiability to systems. [BMR+96]
	Microkernel	Applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and customer-specific parts.
	PAC Presentation Abstract Controller	A structure for interactive software systems in the form of a hierarchy of cooperating agents. Every agent is responsible for a specific aspect of the application's functionality and consists of three components: presentation, abstraction and control. This subdivision separates the human-computer interaction aspects of the agent from its functional core.
	Reflection	A mechanism for dynamically changing structure and behavior of software systems. It supports the modification of fundamental aspects, such as type structures and function call mechanisms. In this pattern, an application is split into two parts: (1) a meta level that provides information about selected system properties and makes the software self-aware; and (2) a base level which includes the application logic.
Design Patterns	Client Dispatcher Server	An intermediate layer between clients and servers, the dispatcher provides location transparency by means of a name service, and hides the details of the connection establishment between clients and servers.
	Decorator	Attach additional responsibilities to an object dynamically. Provides a flexible alternative to subclassing for extending functionality on a per object rather than per class basis. A Decorator (vs. Strategy) lets you change the "skin" of an object.
	Proxy	Provides a surrogate or placeholder for another object to control access to it.
	Publisher-Subscriber	Used to keep the state of cooperating components synchronized by enabling one-way propagation of changes: one publisher notifies any number of subscribers about changes in state.
	Strategy	Supports algorithmic interchangeability (within a family of algorithms) through their encapsulation and allowing selection to vary independently between client instances. A Strategy (vs. Decorator) lets you change the "guts" of an object.
	View Handler	Manages all views that a software system provides via a view handler component that allows clients to open, manipulate and dispose of views. It also coordinates dependencies between views and organizes their updates.

Figure 2.12: Various Architectural and Design Patterns

The spiral methodology can be characterized as iterating over the processes of "think a little, plan a little, implement a little, test a little". Each of the four phases is given equal treatment within each cycle of the spiral, for a given level of detail. The entire process is then repeated at a subsequent level. Therefore, while the "deliverables" from each phase do not change in structure, their content is different. As the methodology progresses from level to level, more and more detail is generated, and after several iterations, it is complete.

As applied to the engineering of a product, the spiral methodology allows for feedback regarding the complexity of a particular requirement. There are stages where mistakes in the requirements can be corrected and the end user gets an early look at the results and provide feedback. The implementation team can offer

feedback regarding performance and viability back to the requirement team and the design team for the next iteration of the spiral. Consequently, the product can track technology better and as new advances are made, they can be incorporated into the architecture.

Similarly, such an approach can be applied to the management of collaborative behaviours. A collaborative system must be able to manage the lifecycle of its behaviours by dynamically examining, computing, modifying and utilizing changes in their characteristics and that of the collaboration being performed. Therefore, the spiral methodology can be seen as a way to express the necessary iterative approach that is part of providing a system which can "engineer itself". This notion is addressed in Chapter Three when the notion of "self-investigative behaviours" is further detailed.

2.1.3.2.4 Beyond Object-Orientation: Responsibility-Driven Organization

One difficulty in the development of software systems is deciding how to organize its functionality as performed by the various software components. Object orientation is a common design and implementation technique that attempts to address this concern by viewing a system as a collection of interacting objects, each of which represents and encapsulates the details of some (real-world) entity. Unfortunately, an object's complexity is often significantly increased as a result because a single entity assuming too many diverse responsibilities. A video-on-demand system, using a movie object with audio and video tracks, would need to address issues such as data retrieval, decoding, intra/inter-stream synchronization, user interface management and so forth. While most modern designs would not attempt a strictly monolithic solution, it is quite common to blend at least some of these aspects together in implementation-specific ways. The result is difficulty in isolating which aspects of the system deal with different parts of the problem, ultimately affecting how easy it is to debug, modify and maintain the system. Levels of functionality are often woven together and blur the distinction between solution, policy and mechanism [CMB+90].

Within collaborative environments, this lack of separation affects the ability to scale and adapt such systems because the range of possible usage scenarios can be overwhelming. Combined with the integration of presentational techniques, live media and interactive exchanges, there is clearly a need to simplify when, where and how to address these different aspects within a system's overall design. In such a situation, object orientation is a necessary but insufficient means of isolating system detail because each object must still address issues relating to its own behaviour as well as its interaction with other objects. The result is a mesh of interacting objects, each communicating with the other in a very tangled manner. In general, objects are both busy doing their own work (i.e. domain computation) and telling others what to do (i.e. coordinating them), either explicitly or implicitly through their direct communication with each other. This leads to a system that is a highly interdependent and interconnected collection of objects which must directly deal with the coordination of other entities in its environment. Consequently, the entities within the space are highly configuration-dependent and their ability to be re-used within a general framework is reduced.

Therefore, there is a need to organize the system along lines of the responsibility in which a system's execution (i.e. domain computation) is separated from its coordination (i.e. the management of domain computation). The coordination *meta level* manages the base-level computation in which these two aspects of an entity are explicitly separated. Doing so facilitates cleaner system design and re-use at a behavioural level in addition to a structural (i.e. functional) level (as in the standard object-oriented paradigm). This takes the standard hierarchical approach to structure (e.g. classes, types, objects) and mirrors it in a behavioural context; that is, lower-level domain activities are managed by higher-level coordination activities.

As applied to collaborative media space design, this approach reflects the difference between the elements within it. Media are synchronized at multiple granularities (objects, frames, actions, groups of...) and in terms of their individual media-specific requirements. Collaborators who use these media must be coordinated according to their behaviours, which are oriented towards achieving a specific collaborative goal. Consequently, collaboration is achieved by coordinating users' behaviours via synchronizing the media they use. Since these tasks deal with different entities and at different levels of abstraction within the media space, each forms a meta level to the one below it and serves as the basis for applying a technique known as *reflection* [Mae87] [Smi85].

2.1.3.2.5 An End-to-End Approach: Executable Models

Another difficulty with real-time system engineering is ensuring the final result matches the initial intent. In other words, does the implementation reflect the design. A lot of approaches to multimedia systems, for example, do not adequately address more than one phase of system's lifecycle; rather, most tend to deal exclusively with either the specification/design, implementation or run-time execution phase. Those software development techniques that do span multiple stages, such as the spiral methodology, however, can often suffer from a "semantic gap" between one stage and the next. The result is an increased potential for errors and unintended discrepancies between the desired design and the final system because of the often manual translation from one phase to the next.

To reduce the likelihood of such discrepancies and to facilitate the use of appropriate abstractions, this work utilizes the notion of *executable models*. Such models allows the specification of system entities, such as collaborative scenarios at the design level which are then executed *directly* without the need for manual translation into an implementation. This approach addresses the lifecycle from design through to run-time execution because the design models themselves *are* the implementation which is then directly executed by the computing machinery. Consequently, by modifying the design, you modify the implementation, which directly impacts the actual run-time execution. Therefore, debugging the implementation cannot introduce discrepancies with the intended design and modifications to the design will be automatically reflected in the run-time execution.

Such an approach is used within this work as the means to define various components and patterns that regulate behaviour within the media space. The benefit is that the design of potentially complex collaborative scenarios

can be modelled at more abstract levels (based on patterns and components) which are then executed directly. By modifying the design of the collaborative scenario as a model, you automatically modify its actual realization. Doing so allows the behavioural patterns to be explicitly extracted and specified as high-level event-driven models which can then drive the execution of the objects within the space. Consequently, the rules governing interaction in the space are not embedded implicitly in code modules but exist independently as executable models.

An important aspect of such an approach is the use of *active objects* to realize elements within the system. Each element, be it a user-level collaborative pattern or a part of the media space system itself, is considered an active object with its own thread of execution. Rather than use passive computational structures (i.e. traditional objects) to handle the system's various operational states, an entity's active nature allows it to perform computations and execute actions to fulfill its role. Changes in behaviour or function are achieved by using the appropriate active entity which "knows how" to perform a particular type of activity (i.e. it is a specific executable model).

Investigation of this feature is based on the use of the ObjecTime [SGME92] [Sel94] CASE package and will be discussed in subsequent chapters.

2.1.3.3 Middleware

Middleware is an approach to software development that evolved during the 1990s in order to promote connectivity between and the development of distributed applications within networked environments. Effectively, middleware offers a set of services between the application and the OS/network layers that enable multiple processes running on different (potentially heterogeneous) machines to interact across a network (see Figure 2.13 [Ber96]). It does so by abstracting and providing augmented/additional functionality to the application with the following benefits [Sch97]:

- *Location Transparency*: A 'holy grail' of distributed processing, middleware attempts to make the locality of the service transparent to both the user and the application itself.

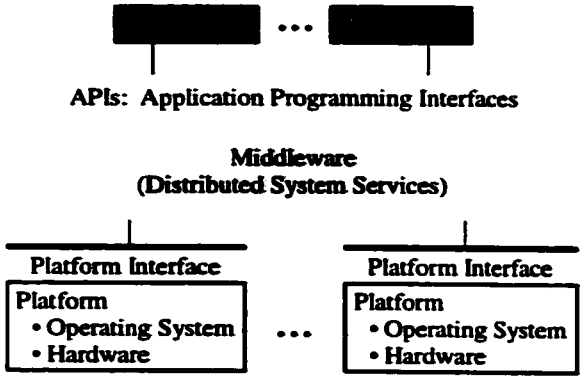


Figure 2.13: The Role of Middleware

- *Network Independence*: Removes dependencies and specifics of the underlying communication services from the application.
- *Reliability and Availability*: Ensuring provision of the service is not the responsibility of the applications but of the service provider (middleware).
- *Scalability*: As above, changes to service support for handling increased load can be addressed separately from the application, which allows better management and evolution of its functionality across all of its users.

Often called "the middle tier" in colloquial terms, the design of middleware usually takes on one of the following forms [Ber96]:

- *Transaction Processing Monitors (TPM)*: TPMs are a means to utilize a collection of middleware services within distributed applications by mapping between these services to the transaction processing approach commonly used by many distributed applications.
- *Remote Procedure Calls*: Commonly known as RPCs, remote procedure calls are a system-level abstraction which allows parts of an application's execution (i.e. procedure invocations) to be made across process and/or machine boundaries.
- *Message-Oriented Middleware (MOM)*: MOM is a basic information transfer paradigm that abstracts and facilitates inter-program data exchange using message passing. Its main benefit is the abstraction of the messaging away from network-specific concerns, thereby promoting interoperability, portability and flexibility within applications.
- *Object Request Brokers*: Object Request Brokers (or ORBs) are a mechanism to manage a program's objects such that they may be distributed and shared across heterogeneous networks. A well known example of this approach is CORBA – the Common Object Request Broker Architecture [YD96]. [Vin97]

Recently, the use of middleware within distributed multimedia systems has been increasingly popular [BCRP98] [CBC98] [BCC+99] [Cou99] [MBK+99] [LN99a] [LN99b]. As a means to provide more abstract network-related services, interest has also been expressed in terms of providing for reflective middleware (Figure 2.14) as structured means to address tailorable network services (see Lancaster publications [BCRP98] [CBC98] [BCC+99] [Cou99]).

The research done as part of this thesis can also be seen as an example of reflective middleware. Furthermore, the MSpace prototype also illustrates the use of another well-known middleware system, the Distributed Component Object Model (or DCOM) [BK96], which will be discussed further in upcoming sections.

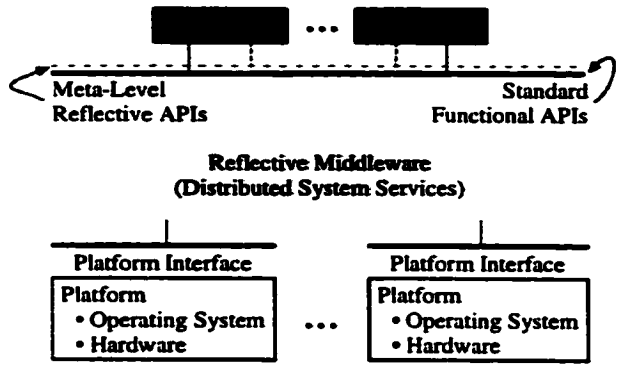


Figure 2.14: Reflective Middleware

2.1.3.4 The Meta Approach

The prefix *meta* denotes a "more comprehensive" yet critically related approach to a specific subject [Web89c]. That is, the meta approach to a discipline uses abstract representations of that subject rather than dealing with specific concrete entities in an underlying system. Consequently, the power of the meta approach lies in its expressiveness and ability to encompass the complexity of its underlying base systems.

To provide background perspective for the terminology used within this work, the topics of metaobjects, metaobject protocols, metacomputing and metamodelling are briefly overviewed.

2.1.3.4.1 Metaobjects and Metaobject Protocols

Simply put, a metaobject is just an object. The differences lies in the level of abstraction and the focus of the object itself. A metaobject is effectively an object at the meta level, relative to a given base level, as shown in Figure 2.15. The domain of the metaobject is some aspect of the base object which is defined by their particular relationship. For example, a given base object is an MPEG codec [BL94], with all the necessary mathematical logic for performing the encoding standard. However, because different variations of the MPEG stream are possible (i.e. different arrangements of I, P and B frames), the actual arrangement could be managed by a metaobject which solely is responsible for dealing with which frame arrangements should be used. Other examples include which virtual memory page replacement policy or CPU scheduling policy are used within an operating system [Yok92] [Yok99].

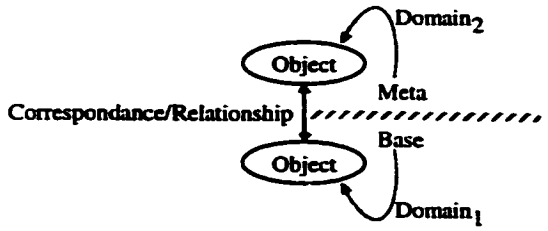


Figure 2.15: Meta and Base Objects

Metaobject protocols [KP96] are the basis of a technique usually associated with programming language design. By offering supplemental interfaces to programming languages, metaobject protocols offer a way for users to incrementally modify a language's behaviour and implementation. Therefore, metaobject protocols allow users to modify the expressiveness of the language and the ability to write programs in it. As part of promoting flexibility, metaobject protocols are based on the use of an object-oriented language (hence the word "object" in "metaobject") as well as the notion of procedural reflection (detailed later in this chapter) by which modification to the language can be done in terms of its own syntax.

The use of metaobject protocols blurs the distinction between language designer and language user. Traditionally, languages were built with well-defined, fixed behaviors (i.e. semantics) in which users treated languages as immutable black-box abstractions, and any needed flexibility or power was derived from constructs built on top of them. This approach was based on the notion that programming language design was difficult, highly-specialized and inappropriate for the average user. In contrast, the metaobject protocol approach is based on the idea that one can and should open languages up, allowing users to adjust the design and implementation to suit their needs. In a language based upon metaobject protocols, the language implementation itself is structured as an object-oriented program and allows the power of object-oriented techniques to be exploited internally. This can be seen as taking less of a "black box" approach and more of a "glass box" [UMAI94] [KP96] approach in which the internals of the system can be seen and addressed in order to meet the needs of specific usage contexts.

The recent trend to "opening up" systems using the concept of *open implementation* is based on the observation that technological systems other than programming languages can be based on a metaobject protocol. Within language design, the effect of opening up its implementation is that it does not represent a single point in the overall space of language design, but rather an entire region within that space. Therefore, any approach such as collaborative system framework that utilizes the metaobject protocol approach can address not just a specific way to collaborate but a naturally expansive way to support a wide range of collaborative practices and their support systems.

2.1.3.4.2 Metacomputing

Metacomputing is "computing about computing". More commonly referred to as *distributed computing*, metacomputing refers to the inherent notion that a distributed system inherently performs computations about its constituent systems in order to distribute the original computation. As such, metacomputing deals with information and abstractions regarding individual computing systems as they exist within a larger, integrated whole.

Inherent in such an approach is the need to coordinate the activities done as part of distributing the computation [RBD+97]. Therefore, the general "management" principle of metacomputing is analogous to coordinating a range of activities used within the collaborative process. In each case, a semantically coherent environment is

provided not only by correct synchronization of individual activities but by their coordination relative to each other and the overall system.

An important part of achieving this organization is the communication and distribution facilities provided by the underlying operating environment. This infrastructure can range from general-purpose distributed operating systems [Bur94] to specialized "application" kernels [Fuj90] [Jef85]. These systems typically provided numerous services including standardized communication methods between hosts. Traditionally, this often took the form of a *Remote Procedure Call* (RPC) [Tan92] [Bur94] in which case the execution of a particular functionality (i.e. procedure) is transparently mapped to a remote machine. This notion has been augmented within the object-oriented paradigm towards a versatile and encompassing mechanism to separate a system's logical functionality from its physical implementation (i.e. location). This middleware approach (as discussed previously) is commonly based on either CORBA, the *Common Object Request Broker Architecture* [Vin97] [YD96], or DCOM, the *Distributed Component Object Model* [BK96] [Gri97]. The role of DCOM within the prototype developed as part of this research will be discussed in upcoming sections.

2.1.3.4.3 Metamodelling

Modelling is a technique used in many disciplines to represent and aid in the understanding of virtually all kinds of systems. The term *metamodelling* refers to principle of abstracting and describing the thought processes behind the development of specific models; that is, the model uses descriptive abstractions rather than actual system entities [GS96]. The use of a higher representation allows a simpler and more abstract model based on behaviours and relationships rather than specific attributes. Consequently, it reduces the complexity of a specific system and focuses on the high-level rather than base-level behaviour. This also provides more flexibility and variability in design such that the model implicitly provides a framework for different possible implementations rather than one particular instance. This approach is taken by this research in order to provide an infrastructure that can be used across a multiplicity of collaborative systems rather than be fixed to a single domain.

2.1.3.4.4 Summarizing the Meta Approach

The meta-level approach offers a general framework by which to consider media space behavioural management as a meta system itself; that is, as a system which coordinates and synchronizes media and user interactions, the media space is itself performing meta-computation. Use of a metaobject protocol enables such a system to provide a mechanism by which the domain functionality of the media space can be "openly" examined and therefore managed in both a coordinated and well-defined manner. Furthermore, a clean and well-defined design is facilitated through employing the object abstraction so that the system can be regarded as a group of communicating objects, whose behaviour can be governed through the use of explicit behavioural metaobjects and their corresponding metaobject protocols.

2.1.3.5 Reflection

Reflection is an increasingly popular yet somewhat esoteric organizational and design paradigm that originated in the fields of artificial intelligence and programming language design. Most easily understood[4] as "... *the process of reasoning about and/or acting upon oneself*" [Mae87], reflection can be thought of as a way for systems to "think" about themselves. In concrete terms, a reflective system is one which computes about itself in addition to computing about its application domain. To do so, it utilizes a two-level computational structure which *causally connects* a domain-specific base level with a corresponding meta level. Specifically, the meta level's computations deal solely with base-level entities, while the base-level entities compute about a specific functional domain. That is, the meta level computes using a *self-representation* of the system while the base level deals with the application's primary functionality.

Reflective architectures can be extremely useful for applications which must initially meet specific requirements but later evolve to comply with new ones, depending on environmental changes. For example, consider an application whose objects are persistent but not replicated. Later, due to a change in policy (e.g. increased availability is required), certain objects must be replicated. In a reflective architecture, persistence and replication can be implemented as meta-level constructs relative to the basic object and its functionality. The ability to *dynamically* associate non-functional requirements to (groups of) objects offers increased transparency and adaptability – two important aspects of both real-time and collaborative systems.

Reflection therefore offers a general approach to abstract and interrelate different levels of a system. It can effectively be thought of as a *pattern* [BMR+96] for structuring system organization into *causally connected levels of responsibility*. While each level performs its own independent activities, it also has a direct correlation to other layers, either as to the information observed or the actions impacted upon them. Therefore, as discussed in subsequent chapters, it serves as a vantage point for the consideration of self-adaptive software systems [Ore96] [OGT+99] [KS99].

2.1.3.5.1 Illustrating Reflection: Debugging A Computer Program

To illustrate the notion of causally connected levels of responsibility and their interaction, consider the process of debugging a computer (application) program (Figure 2.16). In reflective terms, the application itself is the base level which performs subject-specific computation. The meta level is the debugger which computes about the application but does not know or care about what the application actually does. Hence, it has different semantics, level of abstraction, computational domain and complexity. And as illustrated by the fact that a generic debugger can be used with different application programs, a single meta level can potentially deal with many possible base levels. When the debugger examines elements of an executing application, the system is

---

[4] While a reasonably intuitive concept, the nomenclature associated with reflection is sometimes quite obscure. The reader is therefore referred to Appendix B for definitions of the various terms used here and throughout this dissertation.

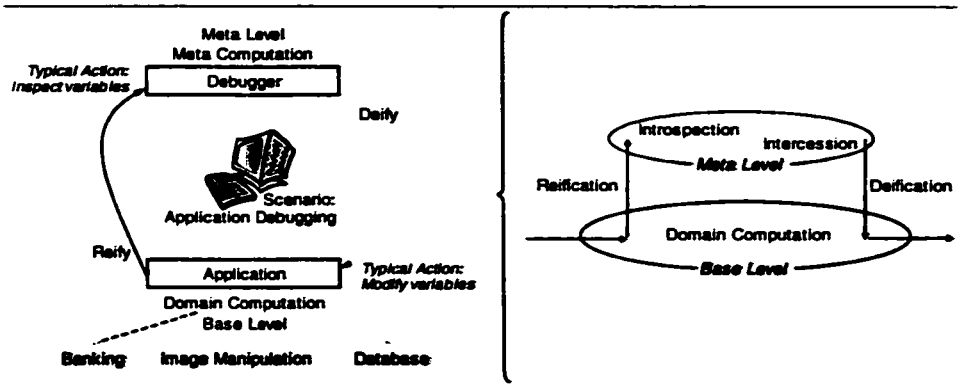


Figure 2.16: Debugging as an Illustration of Reflection

*reify*ing those elements (taking them to a new level and using them as data for computation). When the debugger is used to modify those elements to affect subsequent computations by the application (i.e. elements are modified so as to alter the execution of the application), this is the process of *deification*. While usually only considered a two-level example of reflection, should the debugger be invoked upon itself (say for example, in response to the debugger itself crashing), the resulting three level structure would form an example of a *reflective tower* (Figure 2.17).

In the general case, as illustrated on the right side of Figure 2.16, a system moves from performing base-level computations about a specific domain to its meta level via *reification*, where it can examine aspects about its base level (through *introspection*). It can then make changes to the base level (by applying *intercession*) which are then utilized when the base-level computation resumes (upon *deification*). In an implementational sense, reflection supports flexibility in a "physical" architecture, while in terms of computation, it affords flexibility at a behavioural level. This synergy between a meta-level architecture and reflection is summarized in Figure 2.18.

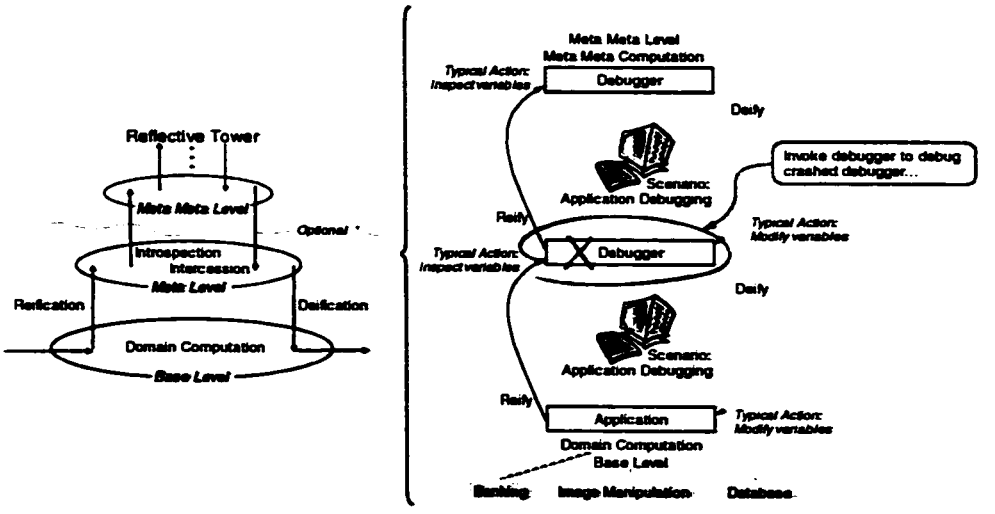


Figure 2.17: An Example of Multiple Meta Levels – The Reflective Tower

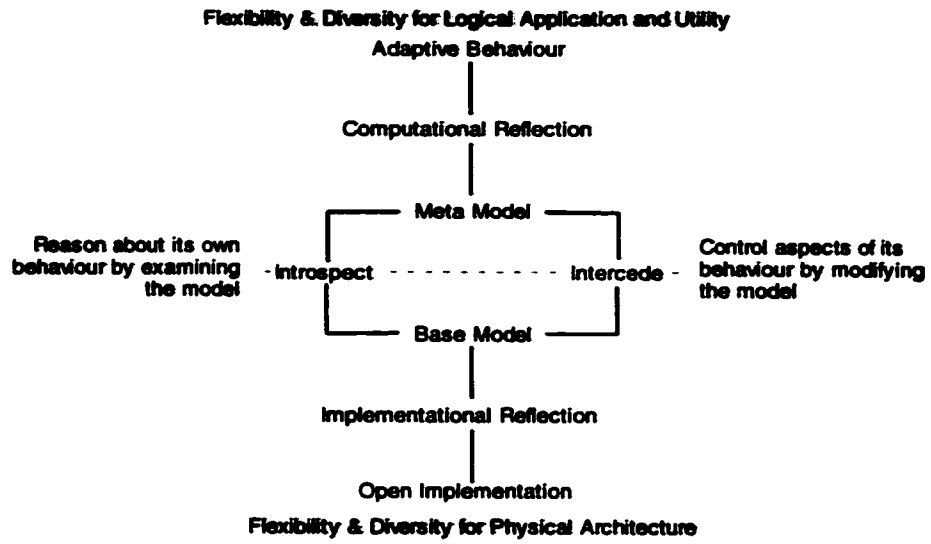


Figure 2.18: Reflection and the Meta Level Architecture

2.2 Implementational Background: Technology Overview

As part of the research done for this dissertation, a proof-of-concept prototype was constructed on the Windows® NT platform. Several complex technologies were used to do so and this section very briefly overviews the most important ones so as to assist the reader in subsequent chapters. There is no attempt to be exhaustive in their explanation, but rather, give a basic outline. The interested reader is referred to the numerous possible references on each of these topics for more detailed information. Further relevant details on each will be given when the prototype is presented in subsequent chapters.

The technologies used in the prototype implementation include the following; their basic relationship is illustrated in Figure 2.19:

- MFC: Microsoft Foundation Classes
- ATL: Active Template Library
- D/COM: The (Distributed) Component Object Model
- OLE: Object Linking and Embedding

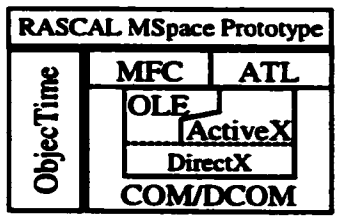


Figure 2.19: Relating Implementational Technologies

- ObjecTime Developer
- ActiveX
- DirectX

2.2.1 MFC, ATL and the Developer Studio

As part of simplifying system development for Windows, two class library techniques are offered by Developer Studio, the C++ development environment used as part of this project. These class libraries, namely MFC [Bla97] [Tem97] and ATL [Box97], offer higher level functionality which encapsulates parts of the low-level Win32 API. The basic difference between them is their focus and the relative complexity associated with certain kinds of programming activities. While detailed later, these two techniques offer the following benefits:

- MFC: provides support for the document/view architecture that supports compound documents as utilized by OLE which is used for the prototype's "integrationist" interface
- ATL: offers better support than MFC for the many complexities associated with using COM, DCOM and the development of ActiveX controls

2.2.2 D/COM: The (Distributed) Component Object Model

The Component Object Model [COM95], commonly known as COM, is a component integration framework developed by Microsoft as part of their Windows® operating system strategy. Both as a specification and an implementation, COM allows developers to build systems out of reusable components (potentially from different vendors) which communicate via the underlying COM subsystem. By defining an API that allows the creation of components based on a binary standard, components can interoperate with other such components and be reused across custom COM-compliant applications.

DCOM, or the Distributed Component Object Model [BK96] [Gri97], is an extension to the Component Object Model that allows network-based component interaction. Effectively, DCOM is the distributed (or network-capable) version of COM; that is, while COM's functionality is limited to a single machine, DCOM supports distributed objects across a network. With DCOM, components operating on a variety of platforms can interact, as long as DCOM is available within the particular operating environment. Consequently, DCOM is an example of middleware, and it is best to consider COM and DCOM as a single technology[5] that provides a range of services, from component integration on a single platform to component interaction across heterogeneous networks. Within its implementation, COM and DCOM are actually merged into a single run-time subsystem which provides for both local and remote access.

---

[5] Consequently, the terms COM and DCOM are typically (and will be) used interchangeably except where need arises.

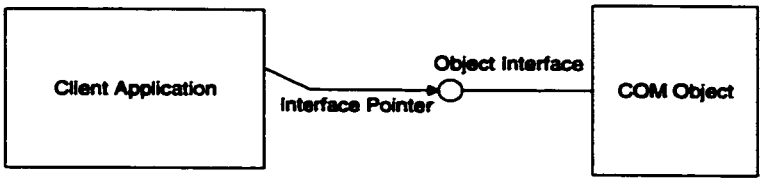


Figure 2.20: Client Application Accessing a COM Object

Services implemented by COM-compliant components (sometimes called COM objects) are exposed through a set of interfaces that represent the only point of contact between clients (programs that utilize the functionality of the object) and the object itself. In fact, one of its 'oddities' is that COM objects are never instantiated or handled by the user. The only part of the object that a user has access to are the object's interfaces, which serve as 'handles' by which the object is manipulated via so called 'interface pointers' (see Figure 2.20). A COM object can support any number of such interfaces, each of which is a logically grouped collection of related methods. For example, Figure 2.21 shows a COM object that emulates a clock. IClock, IAlarm and ITimer are interfaces used to access the functionality of the clock object. IClock can provide the appropriate methods (not shown) to allow setting and reading the current time. Additionally, the IAlarm and ITimer interfaces can supply alarm and stopwatch methods. The IUnknown interface is a standard interface that all objects have as part of the COM standard in order to control the object's lifetime and obtain access/locate to the object's other interfaces. Thus a COM object can be considered (and is sometimes called) a server, as it functions in client/server context for the application utilizing it.

While DCOM offers basic support for distributed objects, there is notably no current support for features such as real-time processing, high reliability and alternate communication paradigms beyond synchronous RPC, such as multicast[6]. These issues will be further discussed as they become relevant to the implementation of the RASCAL MSpace prototype.

2.2.3 OLE: Object Linking and Embedding

While COM and DCOM represent low-level component technologies, Object Linking and Embedding, more commonly known as OLE [Bro95], is a higher-level, application services platform built on top of COM. Developed at Microsoft in the early 1990s, OLE is a compound document[7] technology whose basic premise is to allow one application's data to be inserted as an object within other application's documents. When this object needs to be worked with, the application that understands (i.e. used to create/edit/view) that object is invoked automatically. The object within the document can be either embedded (a fixed, internal version of the object) or linked (referring to "live" version of the object in a separately existing file); see Figure 2.22.

---

[6] At the time of this writing, the emergence of COM+, the "second generation" of COM/DCOM to provide enhanced functionality may address some of these issues; others, however, remain topics of ongoing research.

[7] Compound documents are those generated from multiple source tools.

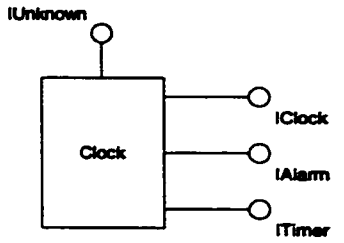


Figure 2.21: COM Object With Multiple Interfaces

The fundamental tenet of OLE is the integration of information in a document-centric rather than an application-centric manner. That is, the idea behind OLE was to have the (operating) system to coordinate application-to-data mapping so that the user could simply just "work" without dealing with the organization of the tools involved. Hence, the user interacts with a document and the document "magically" manages its contents. In effect, such compound documents act as "containers" and OLE objects are "contained" within them. In this way, the document transparently handles the semantics and syntax of data "behind the scenes".

This use of a compound container document is analogous to a media space in terms of functioning as an integration framework. For this reason, combined with its component-based nature, user interface integration features (such as in-place editing [Bro95]) and permeation on the development platform, OLE seemed a suitable candidate to illustrate the media space concept. Several key issues important to OLE, however, became of concern as development progressed; they are briefly mentioned here and further explored during discussion of the MSpace prototype. They are:

- *Data Associativity*: how to relate data to the appropriate application tool
- *User Interface Integration*: how to manage the user interfaces of different tools so as to focus on manipulation of the data and not management of the tools
- *User Interface Liveness*: how changes in data are reflected in the user
- *Event Manipulation*: how to notice, intercept, respond to and effect application events and changes in data programmatically

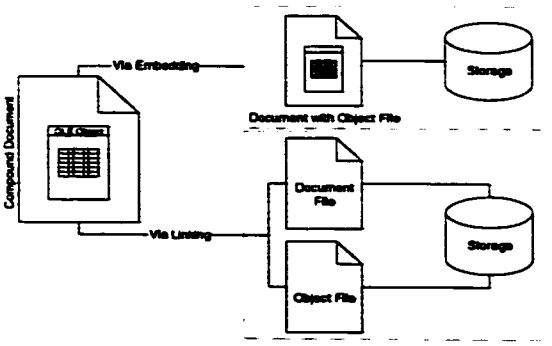


Figure 2.22: OLE Compound Document Alternatives

Unfortunately, a significant deficiency of OLE in terms of real-time applications is its use of static server representations when an object is not the focus of user input (known as "in-place active" in OLE parlance). Specifically, there is no standard means to handle modern real-time "continuous sourced" entities (e.g. video) and their user interface *in all cases*. This issue of "user interface liveness" will be discussed much further in subsequent sections and will be shown to be very problematic in achieving a satisfactory implementation.

2.2.4 ObjecTime: Real-Time Object-Oriented Modelling

ObjecTime Developer [SGME92] [Sel93] [Sel94] [STD94] from ObjecTime Ltd. of Ottawa[8] is a computer-assisted software engineering (CASE) tool for the development of real-time distributed systems. Based on the Real-Time Object Oriented Modelling (or ROOM) methodology [SGME92], ObjecTime offers facilities for the modelling, simulation and analysis of real-time distributed communication systems. Additionally, ObjecTime also functions as a prototype development tool in which models developed within the tool can be taken from simulation through to code generation of actual running systems. Consequently, ObjecTime is an example of a tool which offers support for "executable models" which were discussed earlier. Additionally, it takes an event-driven, object-oriented approach to systems design which is amenable to the nature of media-based and interactive systems undertaken as part of this research.

Based on a visual design metaphor, ObjecTime models are "drawn" according to the ROOM structural diagramming conventions. In employing an object-oriented approach, ROOM uses classes orthogonally across definition of all key system components, ranging from actors to protocol definition, as well as the classic datum interpretation. These classes serve as the means by which the system designer/architect constructs his/her model(s): (1) *actor classes*, which define the (computationally) active objects (i.e. actors) which independently carry out specific responsibilities within the system; (2) *protocol classes*, which specify how actors communicate; and (3) *data classes*, which specify how generic information/data is represented. Actors can be created statically or dynamically, in single or multiple instances and imported (i.e. "inserted") into role-specific placeholder "slots" as required. Each actor runs independently, can contain other actors using structural containment (i.e. nesting) and can communicate with other actors through the exchange of messages. These messages are defined as instances of a specific protocol class which are sent over connections between ports of a given protocol class. These messages (which constitute incoming events for a receiving actor) then trigger changes in that actor's behaviour, which is specified using a (hierarchical[9]) finite state machine. The resulting model is then directly executable within the ObjecTime toolset (via simulation) or can be compiled into executable software using ObjectTime's code generation facilitiates.

---

[8] During this work, ObjecTime Ltd., was wholly acquired by Rational Software, Inc. Their technologies, (Rational Rose, ObjecTime Developer) will be combined into a "single comprehensive UML engineering tool" called Rational Rose RealTime. Existing ObjecTime models will need to be migrated to the new tool.

[9] Unless otherwise noted, reference to "finite state machine" can be considered synonymous to the "hierarchical finite state machine" as utilized by ObjecTime.

As part of interacting with a larger environment, ObjecTime allows the integration of existing code modules within its models; therefore, it is generally amenable to the creation of a simulation-based prototype. Furthermore, since it is able to generate code for a number of real-time operating systems, use of ObjecTime lends itself to future projects that could arise from this research. However, while matching the domain of the thesis work, some difficulties encountered with use ObjecTime include its very steep learning curve, non-reflective and somewhat closed meta-level architecture, and the difficulty of integrating it with development technologies like that of MFC/ATL, D/COM and OLE. These issues, along with the basics of the ROOM diagramming technique and ObjecTime's class system will be further discussed in the section on the MSpace prototype implementation.

2.2.5 "X"-Technologies: ActiveX and DirectX

Related more by marketing hype than technology, these two "product designations" are not directly related in their functionality. They are only briefly mentioned here so that the reader will be acquainted with the terminology as it becomes relevant. Benefits and issues related to their use will be presented within the discussion of the implementation itself.

ActiveX [Cha96] is (currently) the "global" product family designation for Microsoft's web-based component strategy. Such components are called *ActiveX controls*, and allow specific functionality to be embedded in programs that use or "host" the controls. An example is a web browser which wants to utilize charting facilities whose functionality is offered by a given "charting control". Other examples, as employed in the MSpace prototype, include the Windows Media Player and the WinNov video capture tool.

DirectX [MSC00] is a family of Microsoft technologies for high-performance multimedia. In particular, DirectX has subset technologies specifically for video, graphics, audio and the like. Their use within this research is only in support of using ActiveX controls such as the Windows Media Player.

2.3 Summary

This chapter has overviewed the numerous background concepts on which this research is based. The technologies used in the prototype implementation were very briefly examined and serve as the basis for further discussion of the theoretical framework and its trial implementation presented in subsequent chapters.

3 CHAPTER THREE

RASCAL: THE REFLECTIVELY ADAPTIVE SYNCHRONOUS COORDINATION ARCHITECTURAL FRAMEWORK

The Reflectively Adaptive Synchronous Coordination Architectural framework, more affectionately known as RASCAL, is a reflective middleware approach to the structural and behavioural management of real-time synchronous collaborative media spaces. The model offers a framework and methodology for building systems which integrate behavioural management facilities through the use of an intelligent policy-driven communication subsystem. The result is coordination-based collaborative multimedia support for building semantically-coherent shared media spaces which blend the utility and requirements of presentational, conversational and interactive multimedia.

This chapter presents the underlying rationale, main principles and basic elements that constitute the framework developed as part of this research. Subsequent chapters will detail its application to various collaborative multimedia scenarios as well as describe the conceptual prototype known as MSpace.

3.1 The Motivation for an Adaptive Collaborative Framework

Collaboration is an esoteric phenomenon that can be difficult to describe simply because it is so broad and encompassing. Within the bounds of what it means to collaborate, it can be hard to express where to start, where to end and how to progress from one step to another. Furthermore, because collaboration is not a fixed, but a dynamic activity that changes over time and from instance to instance, it can be seen as a *co-adaptive* [Mac91] and *evolutionary* process. The development of such a process, however, including what is produced, how it is produced and how it evolves and ultimately relates to the intended goal, can be very dynamic and difficult to articulate.

Collaboration often involves taking well-known techniques and solution strategies, and working towards a common goal, combining them in a manner which meets their needs, requirements and available resources.. Consequently, the conventional approach is for collaborative applications and frameworks to provide specific software tools that have "useful" functionality for common collaborative scenarios. For example, an e-commerce system utilizing a virtual shopping mall interface would provide specific retail services ranging from merchandise browsing, clerk/customer interaction to financial transactions. While the success of a system is in how it effectively is used, usage is often considered secondary to functionality. While often ignored in the past, recent emphasis on user interface issues has begun to address end-user usability. A separate issue, however, is usability in terms of behavioural management, which is necessary in order to address the semantic diversity and variation of activity within systems. As shown in Figure 3.1, reaching the desired end requires both the means and the opportunity; in other words, to achieve the desired collaborative goal requires use of the right tools

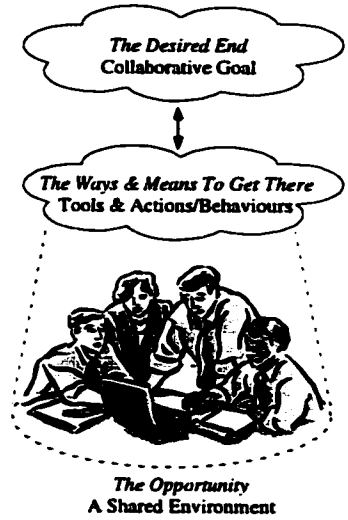


Figure 3.1: The Means and Opportunity to a Desired End

along with the right tasks (such as in the construction example in Figure 3.2). The actions performed constitute entity behaviours (such as collaborative participants/users, tools, applications, ...) within the shared environment. As such, they must meet the needs and intentions of the collaborative process. And because collaboration is a dynamic and evolutionary process, its constituent behaviours must be able to adapt and evolve in order to suit the changing reality of the collaboration itself.

3.1.1 Being Explicit About Behaviour

The notion of behaviour is a common part of daily life, typically involving individuals "doing their own thing", conducting themselves in thought and action according to how and what they perceive both inside and outside themselves. Therefore, behaviour can be seen as intrinsic to its exerciser as well as extrinsic to those affected by its external realization. It is this disjoint perception of what is sensed externally versus what perceived

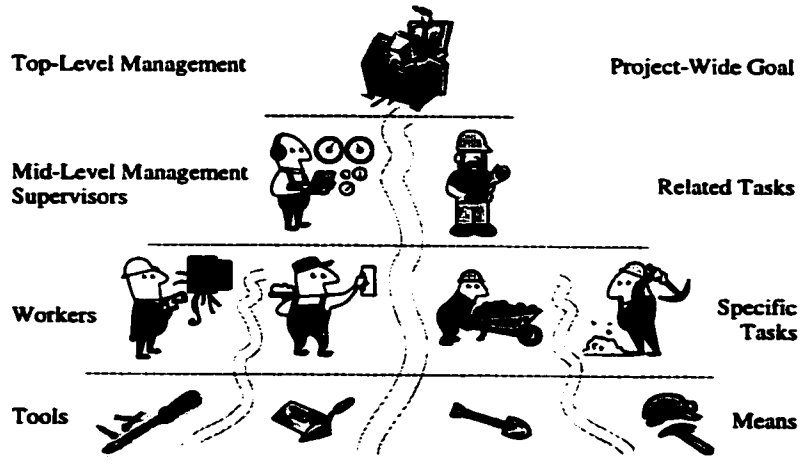


Figure 3.2: A Construction Example: Goal, Tasks and Tools

internally that can lead to difficulties, disagreements and related issues between people. In terms of inter-personal behaviour, only when a person "steps back" and examines their own actions can they realize the ramifications of their behaviour and affect a change in both themselves and their interactions with others.

While readily understood in a human sense, the previous illustration of behaviour is somewhat less intuitive when applied to computational systems. As found in Webster's dictionary, behaviour is defined as "the manner of conducting oneself; anything that an organism does involving action and response to stimulation; the response of an individual , group or species to its environment; the way in which something (as a machine) behaves" [Web89d]. Therefore, as a person performs actions in response to events occurring around him or her, so does a computational system react to stimuli from its environment. This "reactive" paradigm forms the basis of real-time systems, which function as event-response systems that execute actions in direct consequence to an event experienced by the system (be it from the external environment or internally generated). Within the real-time domain, the term "behaviour" is commonly used to denote the actions performed as part of the event-response mechanism.

Therefore, in an anthropomorphic sense, real-time systems can be seen to provide a *behavioural approach* to computation. As a well-known example of the real-time domain, multimedia systems exemplify the applicability of behaviour: (1) their event-based, reactive nature realistically mimics the style of human users by providing a timely and responsive interaction paradigm; and (2) in contrast to traditional data processing, real-time systems execute their event-response behaviours as an explicit part of their life cycle rather than an implicit part of a processing flow. In other words, behaviour within a real-time system is paramount.

It could therefore be considered odd that many multimedia systems focus on function rather than behaviour. This bias is easily be seen in systems which offer diverse functionality but very little support in dealing with how those functions are coordinated and managed. An example is the MBONE [Eri94] application suite which offers no coordinated interaction between its applications. Other examples include presentational multimedia systems which embed usage (and hence behavioural) assumptions in their design such that they deal exclusively with select media types or offer only limited, special-purpose functionality.

This singularity in focus and the lack of consideration relative to entity interaction (within a computational system) mirrors the "interpersonal difficulties" that can arise due to the intrinsic/extrinsic perception of human behaviours. However, the ability of humans to "step back" and assess the relationships between themselves and make changes to their interactions with others is a key aspect to how people regulate their lives, both as individuals, as well as in groups such as couples, families, neighbourhoods, communities and so on. This "step-back" to understand what is going on between people is the act of "reflecting" [Web89e] on a person's inter-personal interactions relative to their role in a larger social context. Therefore, similar flexibility can be afforded computational systems, through the ability to examine, compute and affect changes to itself based on the desired outcome and its current state. Considering computational systems (including real-time multimedia systems) in such a manner is therefore a natural application of the technique known as *reflection*, which forms the basis for RASCAL's *self-adaptive behavioural management* approach to collaborative media space design.

3.1.2 Collaborative Dynamism: The Essence of Self-Adaptive Behaviour

The collaborative process involves people using tools to work together in different circumstances. Within a collaborative effort, people may use tools separately, together and/or in a given combination. This combination may stay the same or evolve over time. As it does, the requirements of the collaborative interaction changes in terms of system resources, coordination and synchronization to meet its new semantics. Therefore, the system must be able to realize these changes are occurring and adapt itself accordingly.

For example, consider a group of 10 collaborative participants (users) located within a shared environment. User A is interacting with user B over a shared drawing object; simultaneously, but in a different part of the space, user C is commenting (narrating) on the rendering of a synthetic movie object to a group of other participants in the space. Because the two activity groupings are performing their tasks separately in a independent manner, they are *semantically disjoint* because they are not working together (at that moment in time). Therefore, they have no *semantic dependence* (i.e. they could care less what is happening in the other group). Consequently, attempting to keep a correspondence between these two groups is an unnecessary waste of effort and resources in trying to provide a "connection" between the groups when one is not actually required. On the other hand, knowing who is part of the respective groups is necessary in order to properly maintain their relationships. Hence, the system must be able to dynamically manage who is part of specific behavioural groupings in order to provide the necessary and appropriate behavioural monitoring and control to meet the needs of its users. For example, if the one of the video viewers changed his *collaborative focus* and stopped watching the video and "joined" the group editing activity, the system must detect this and remove his membership from the "movie group" to the "editing group". This in turn may result in further changes to the collaborative environment in terms of resource allocation to meet the changing reality of the activities being performed within it.

Therefore, as shown in Figure 3.3, the collaborative process can be seen as a direct correspondence between the users' collaborative activities and the collaborative goal (result) as facilitated via specific behaviours and tools.

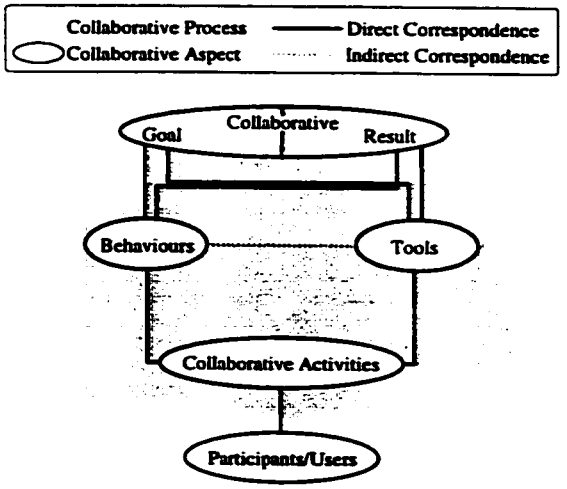


Figure 3.3: Correspondence Between Collaborative Aspects

Importantly, the relationship between the tools and behaviours is indirect as there may be many ways to use different tools to accomplish the same tasks, while on the other hand, the relationship between them directly affects the outcome of the collaborative effort. Therefore, they must be managed carefully. It is important to note that such shifts in collaborative focus and semantics are not something that can be specified totally in advance. Much like the continuous spell-check facilities that examine a word processing document as it is being typed, so must the behavioural management and support facilities of a collaborative environment continuously monitor and adapt themselves to the changing realities of their usage. However, unlike a more narrowly defined set of words within a language, the set of behavioural possibilities can be quite diverse, dynamically evolving in possibly predictable but not necessarily completely predetermined ways. As stated before, this co-adaptive and evolutionary nature of collaboration results in behaviours that can metaphorically be seen as to "take on a life of their own".

This co-adaptive nature of the collaborative process, therefore suggests that a collaborative support system should be more than strictly adaptive, which implies (or can imply) changes within a system occur in response to instructions received from outside the system. Because the system must adapt along side the process it is supporting, there is the need to have the system being able to adapt itself. This *self-adaptive software* approach can be provided to different degrees as illustrated in Figure 3.4 and explained in Table 3.1.

In relation to these techniques, [OGT+99] also presents a methodology for their basic architecture and understanding (Figure 3.5). Shown as two intersecting cycles, the basics of self-adaptivity are broken down into two phases: (1) Adaptation Management (AM); and (2) Evolution Management (EM). The adaptation management cycle performs the "high-level" observation, analysis, planning and instructional activities while the "low-level" evolution management cycle deals with the actual controlled application of changes resulting from the AM cycle [OGT+99].

A simple multimedia example, would be a video server that drops too many video frames when streaming a movie. Classically, the user would detect this and instruct the system to change its frame rate and/or connection bandwidth via some explicit user interface. Here the system would be adaptive in that it could change the way it performs its functionality; however, it does so according to instructions from another entity (the user in this

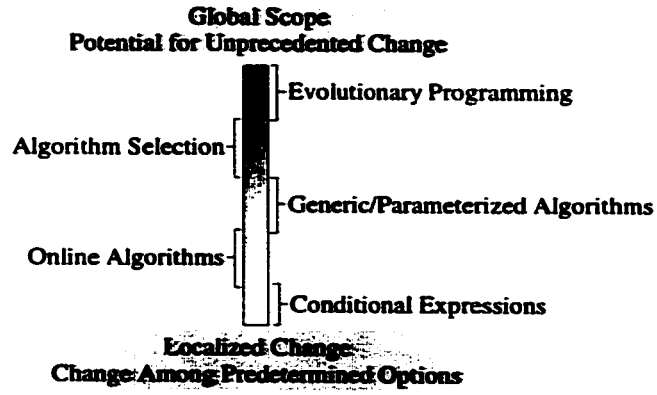


Figure 3.4: Self-Adaptive Software Spectrum [OGT+99]

Adaptation Technique	Explanation	Illustrative Example
Conditional Expressions	Choose program behaviour based on evaluation of conditional (e.g. boolean or relational) expressions.	IF/THEN/ELSE and CASE constructions
Online Algorithms	Full evaluations are periodically mixed with approximation techniques to improve efficiency based on domain-specific knowledge.	Memory paging algorithm (via locality of reference)
Generic/Parameterized Algorithms	Behaviours are customized based on parameter type or input.	C++ template-based classes
Algorithm Selection	Selection is made from a pre-determined set of algorithms made based on environmental conditions.	Dynamic optimizing compilers
Evolutionary Programming	Properties of the operating environment and previous algorithmic runs form the basis of automatically generating new algorithms	Machine learning

Table 3.1: Software Adaptation Techniques

case). A self-adaptive system would utilize the system itself to detect the excessive frame dropping and adjust itself accordingly.

In this simplistic example, the internal logic is not extremely complicated and can be easily seen to boil down to the following sequence:

```
For duration of medium
{
      Monitor frame drop frequency
      If (frame drop frequency > acceptable threshold)
            Perform Adjustment
}
```

Each of these steps is very pseudo code-like and at a very high level of abstraction. The "monitor" step effectively refers to gathering playback statistics (e.g. number of frames played, number of frames skipped, ...). Based on this information, appropriate adjustments can be made relative to how bad the problem is (i.e. the actual frame drop ratio), the resources available to the system (e.g. is one solution more beneficial to general

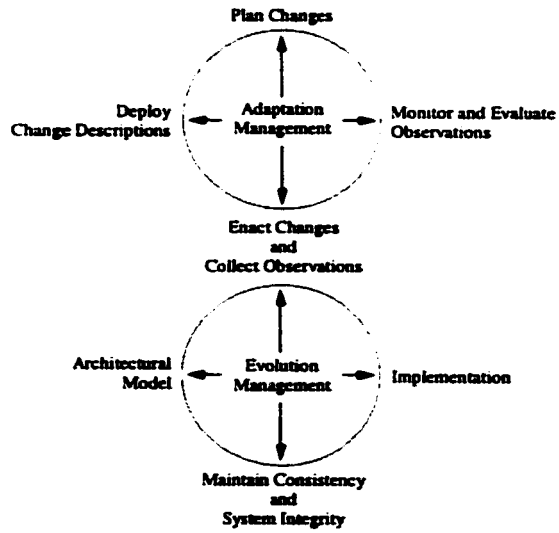


Figure 3.5: AM/EM Software Adaptivity Methodology [OGT+99]

system operation) and/or other system state considerations (i.e. the ability of the system to perform certain actions depending on what else it is doing at the time, such as operating mode). The actual "adjustment" in this case could be a simple choice between two preset playback rates based on a numerical comparison (a single "if/then/else") or a complex mix of frame rate reduction and changes to the connection's bandwidth.

3.1.3 Partitioning for Adaptation: The Need to Separate Concerns

While the preceding situation is reasonably straightforward, the general case may involve many considerations to determine the "best" solution. By taking the standard approach, this logic (however simple or complex) would be embedded within the functional unit's own code. This lack of separation both complicates the functional code by embedding meta-level control logic within it (mixing semantic and abstraction levels) and makes changes to it more difficult in terms of run-time management and evolution. If the logic to determine the appropriate adjustment as well as to perform the adjustments themselves could be isolated, then each could be modified independently of the functionality of the entity in question (Figure 3.6).

Through the use of reflection, different types of adaptation techniques can be applied a using meta-level architecture to define a structured means of controlling various system components. The result is a system which computes at a meta level about its operating conditions while instructing its base level to realize the new conditions (via some functionality or behaviour). Therefore, the AM/EM self-adaptivity methodology can be mapped to reflective terminology, as shown in Figure 3.7. This figure also illustrates these two viewpoints relative to two coarse activities: (1) *self-investigation*, in which a system monitors, analyzes and plans about itself; and (2), *self-regulation*, in which a system alters itself to meet the current specification/requirements. Ensuring this conformance can range from choosing from existing options to adjusting existing infrastructural elements or creating new ones.

In RASCAL, this separation of concerns maps to the use of two "spheres of influence": the *coordination sphere* and the *execution sphere* (Figure 3.8), which are related according to the above meta-base relationship.

The coordination sphere is so-called because of its role as a system-wide meta-level, which in like coordination, can be characterized as relating to "arrangement" for "the harmonious functioning of parts for most effective

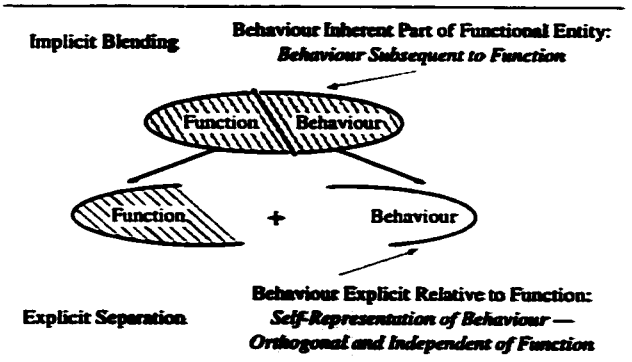


Figure 3.6: Entity Function vs. Behaviour

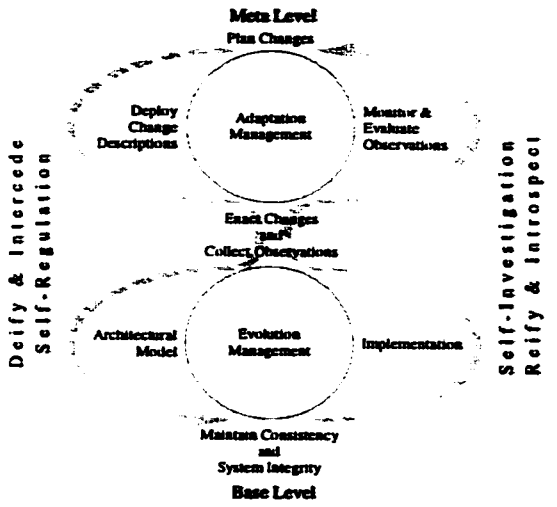


Figure 3.7: Self-Adaptivity from a Reflective Viewpoint

results" [Web89f]. Consequently, the inclusion of synchronization and other management functions common to any system (be it technical or otherwise) can be seen to lie within such an interpretation.

The execution sphere is likewise labelled, although in a somewhat more anthropomorphic sense. Rather than strictly refer to the "computational execution" of a software process, the execution sphere deals with the general notion of "functional execution". That is, the execution sphere deals with "the act or mode or result of performance" [Web89g]. Therefore, in relation to the coordination sphere, it functions as the system's base level whose job is "to do what is provided or required by" [Web89h] the coordination meta-level.

By pairing the two previous system perspectives, a model for *Adaptive System Management* emerges in which the coordination sphere maps to the Adaptation Management cycle and the execution sphere maps to the Evolution Management cycle (shown in Figure 3.9). As such, system management is addressed beginning from the functional domain through to behavioural management. In this minimal form, the coordination sphere deals with the system's behavioural management akin to the AM cycle, and the execution sphere deals with the systems functional domain akin to the EM cycle. However, by taking a reflective view of the system in which

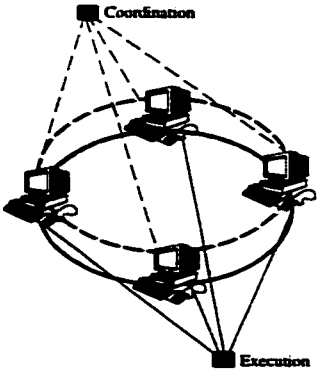


Figure 3.8: Spheres of Influence: Execution vs. Coordination

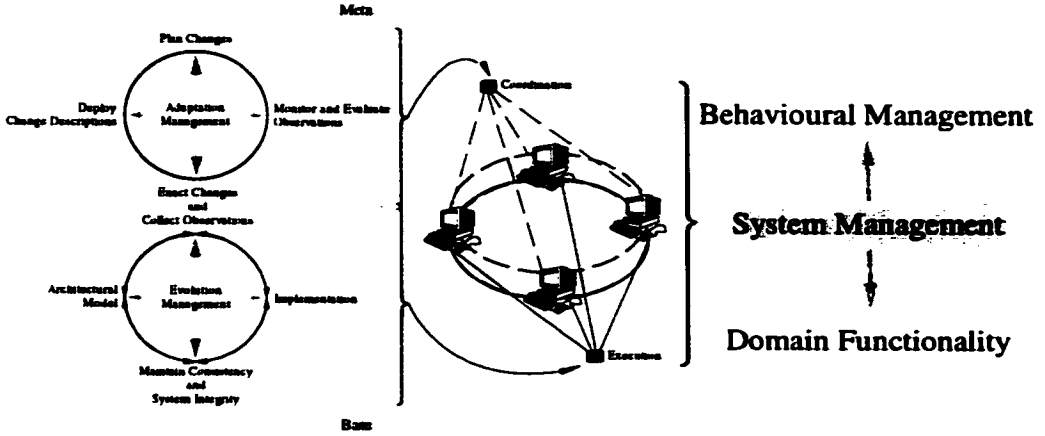


Figure 3.9: Adaptive System Management

there is a *self-representation of both function and behaviour*, the same mapping can be done orthogonally to both coordination and execution spheres. The result is treatment of both function and behaviour as orthogonally self-adaptive subsystems (Figure 3.10).

After partitioning a system's behavioural management into an independent self-adaptive subsystem, the AM/EM methodology can then be mapped to the behavioural management hierarchy as shown in Figure 3.11. Introduced previously, the hierarchy of behavioural management ranges from low-level synchronization at algorithmic and mechanistic level to high-level coordination issues. By comparing these two perspectives, the notion of Adaptation Management appropriately maps to coordination issues related to policy and semantics. Conversely, the EM cycle addresses synchronization which deals with achieving the "intentions" of the AM level at an implementation level.

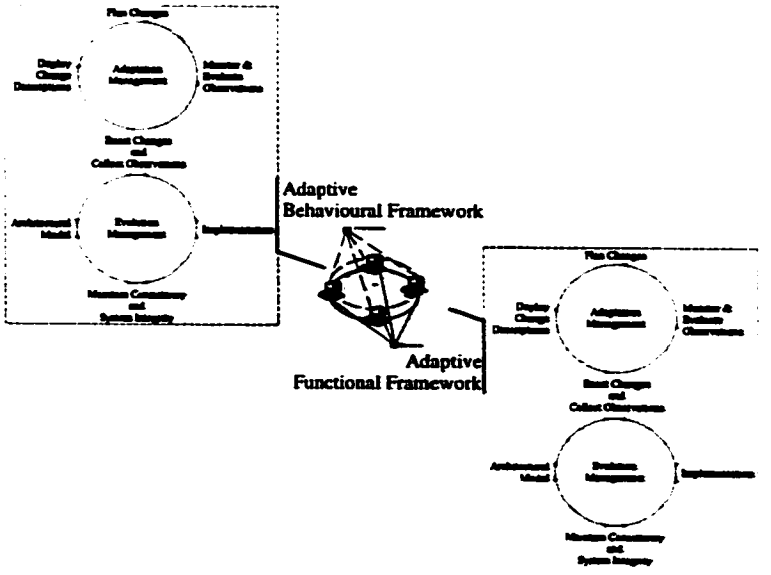


Figure 3.10: Mutual Self-Adaptation — Behaviour and Function

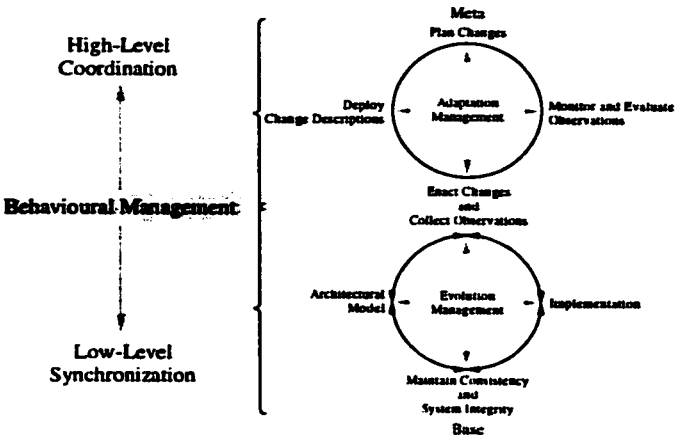


Figure 3.11: Adaptive Behavioural Management — Synchronization to Coordination

Characterized as a meta-level hierarchy, behavioural management can be seen to span from a high-level coordination level is meta relative to a low-level synchronization base level. Such a mapping, therefore, can be extended using the notion of a reflective tower to form the basis for a *reflectively self-adaptive behaviour management hierarchy* (Figure 3.12). The ability to specify behaviours in terms of more abstract, causally connected behaviours, allows for "infinitely" complex behavioural definitions to be built through a combination of behavioural modification and behavioural compositing (i.e. forming more complex and compound behaviours from pre-existing ones).

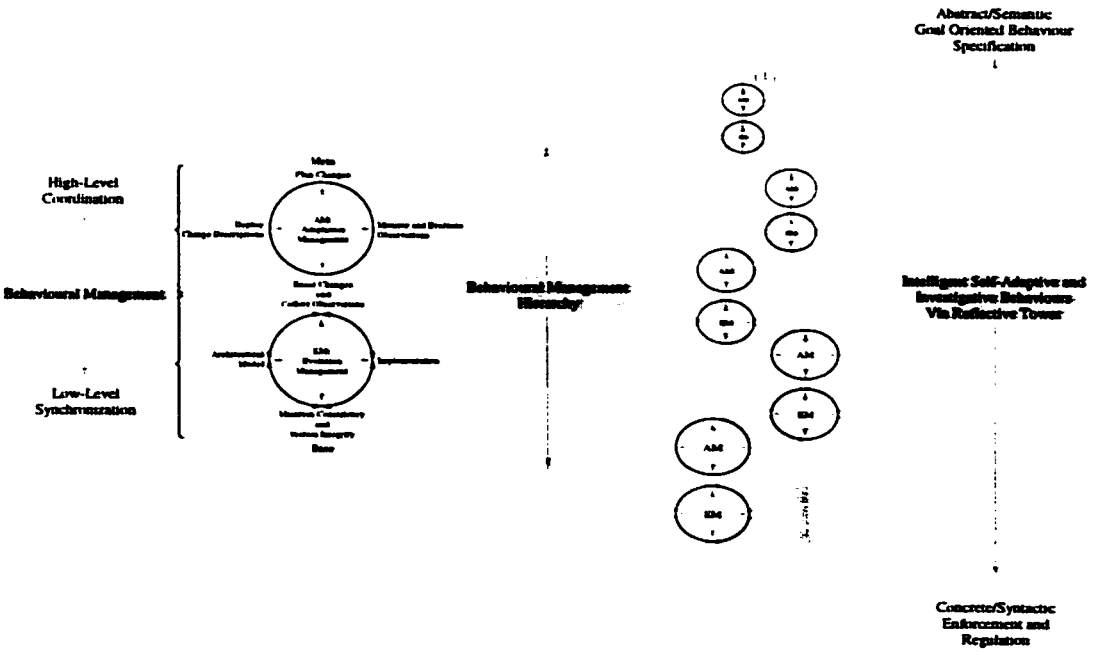


Figure 3.12: Self-Adaptive Behaviour Management Hierarchy via Reflective Tower

3.1.4 Behavioural Building Blocks: Components and Patterns

As introduced in Chapter Two, the use of component-based systems and pattern-oriented software architectures are seeing increased usage within software engineering. Commonly understood in a structural sense, components served as a means of promoting functional re-use while patterns offered re-use of standardized solution strategies.

As a way to organize the collaborative process, the behavioural view constitutes a logical perspective of entities and their interaction. The interaction between users and the media they use is a highly purposeful dialogue in which user activities and the exchanges between them are part of a mutually understood "goal-oriented" behaviour. Therefore, as mentioned in the introduction, participants can be seen to engage in specific collaborative *behavioural patterns* which guide their mutual understanding and activities, both as individuals as well as members of a collaborating group. Complex behaviours are a combination of simpler ones, with each behaviour being a pattern of entities which interact in a specific well-defined manner. The result is a composition of complex patterns from simpler ones, functioning much like *behavioural components* in the overall collaborative effort.

Each component being an independently developed and tested entity with well-defined interfaces to connect to and serve as part of a larger system goes beyond the structural view to also suit the notion of encapsulating behaviours, which also occur in different scales and with different numbers of users and usage contexts. Moreover, patterns of such components offer a broader way to share common behaviours to regulate activities in much the same way the classical pattern offer a way to re-use a general solution methodology across a variety of specific problems.

Therefore, the above compositing approach to behavioural definition is well-suited to the component/pattern approach in which behaviours are regarded as patterns of simpler behavioural components. As such a behavioural pattern may itself be used as a component in another pattern, the result is the recursive behavioural definition shown in Figure 3.13.

The key benefit of such an approach is the customization or *tailorability* offered to system designers, implementers and users. The different elements of the system can be integrated according to the requirements at a range of granularities based on behavioural patterns (i.e. perform a required function, meet specific performance levels, work at a specific level of abstraction, etc.). Patterns are used to specify policy and

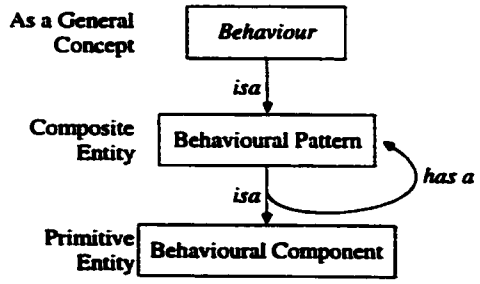


Figure 3.13: Behavioural Definition

collaborative rules as first-class entities that are instantiated and manipulated. Rather than being an inherent part of the structural model, activity coordination involves use of behavioural meta-objects in which the application of specific behavioural rules or characteristics (such as synchronization paradigm) would be realized by using specific behavioural patterns.

3.1.5 Designer Behaviour: An End-to-End Approach

Despite the utility of accepted software engineering practices based on object-orientation, components and patterns, one important requirement is ensuring the final implemented system actually matches the initial design intent. The often introduced "semantic gap" between stages of development results in increased potential for errors and unintended discrepancies. When applied to the design and implementation of behaviours, such inconsistencies can result in confusion since tools may have the required functionality but respond to user commands and interact with other entities in ways that were not intended.

To reduce the likelihood of such discrepancies, *executable models* are used to specify system entities, such as the collaborative scenario, at the design level. These design models are then executed *directly* without the need for manual translation into an implementation. The development lifecycle of the given entity is addressed from design through to run-time execution because the design models themselves *are* the implementation which is directly executed by the computing machinery. Consequently, by modifying the design, you modify the implementation, which directly impacts the actual run-time execution. Therefore, debugging the implementation cannot introduce discrepancies with the intended design and modifications to the design will be automatically reflected in the run-time execution. As a result, the principle of self-adaptivity is simplified by removing the need to maintain consistency between design and implementation as the system dynamically evolves throughout its use (see Figure 3.14).

Executable models also facilitates a higher level of abstraction by allowing the creation of (collaborative) behaviours as models of entities (its topology) and their interaction logic (its own internal behaviour). Shown in Figure 3.15, this means that affecting a change in such an entity (via reflection) is equivalent to modifying its underlying model. The benefit is that the design of potentially complex collaborative scenarios can be modelled at more abstract levels (of patterns and components) which are then executed directly. By modifying the design of the collaborative scenario as a model, you automatically modify its actual realization. Doing so allows the behavioural patterns to be explicitly extracted and specified as high-level event-driven models which can then drive the execution of the objects within the space. Consequently, the rules governing interaction in the space are not embedded implicitly in code modules but exist independently as executable models.

An important aspect of such an approach is the use of *active objects* to realize elements within the system. Each element, be it a collaborative behavioural pattern or a "physical" part of the media space itself, is considered an active object with its own thread of execution. Rather than use passive computational structures (i.e. traditional objects) to handle the system's various operational states, an entity's active nature allows it to perform computations and execute actions to fulfill its role. Changes in behaviour or function are achieved by

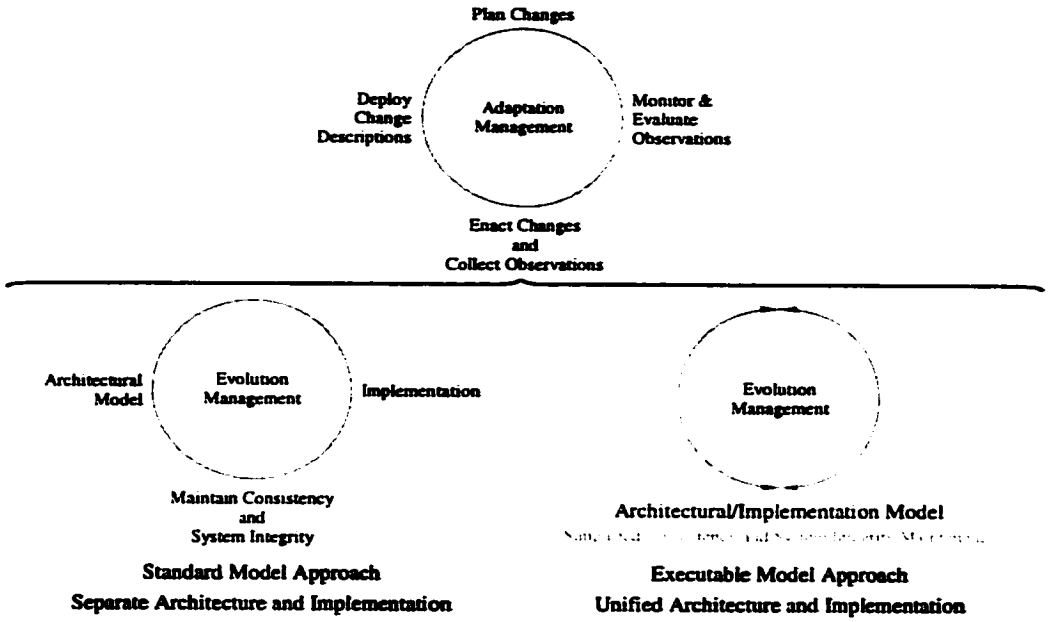


Figure 3.14: Executable Model Approach to Self-Adaptivity

using the appropriate active entity which "knows how" to perform a particular type of activity (i.e. it is a specific executable model). This idea will be discussed further in upcoming sections.

Therefore, collaborative dynamism and self-adaptivity are well-supported by the separation of concerns and the notion of a reflective architecture. Subsequently, how these principles are cohesively mapped to a collaborative multimedia support infrastructure is presented next.

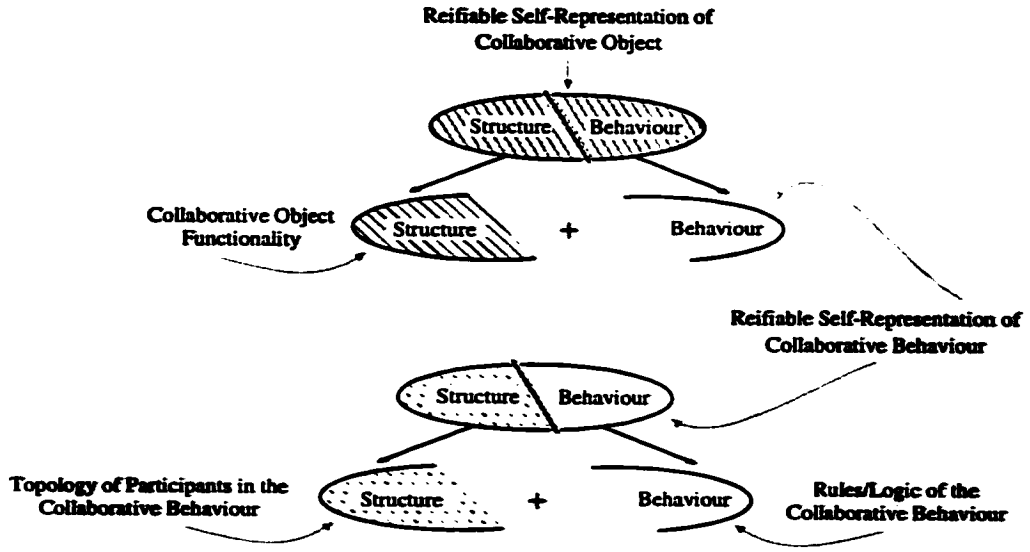


Figure 3.15: Self-Representation of a Collaborative Object and its Behaviour

3.2 The RASCAL Framework

The intent of the RASCAL framework is to provide for the design, deployment and evolution of complex collaborative multimedia systems. The framework offers an object-oriented, component-based methodology which takes an orthogonal view of constructing systems out of both structural and behavioural components. Each component, either structural or behavioural, is a self-adaptive entity which can investigate, regulate and modify itself based on a reflective meta-level relationship to other entities in the system.

This section presents the rationale behind RASCAL, what it stands for and the framework's main elements, including architectural specifics and the reasoning behind them.

3.2.1 The Rationale Behind RASCAL

The essence of RASCAL is the provision for self-adaptive behavioural management. The collaborative process involves people using tools to work together in different ways, and within a collaborative effort, people may use tools separately, together and/or in some given combination. This combination may stay the same or evolve over time. As it does, the requirements of that interaction in terms of system resources, coordination and/or synchronization requirements and indeed its semantics may change. Therefore, the system must be able to realize these changes are occurring and adapt its support for the new situation to suit its resource base and the needs of the system's users as best as possible.

The underlying principle behind RASCAL is that the potential complexity of a collaborative media space is best addressed using straight-forward, independent and scalable abstractions to sufficiently separate system aspects that normally over-complicate its architecture. By separating these different areas of concern, each can manage and adapt itself relative to related sub-systems to achieve a globally coordinated and semantically coherent system that is viable across a broad range of applications. To do so, the technique known as reflection is used to provide a well-defined yet open way of specifying, building, adapting and maintaining a system throughout its life-cycle. It provides a promising infrastructure upon which future enhancements can be incorporated, ranging from user-specified customizations to the creation of self-modifying, intelligent learning-based collaborative media environments.

3.2.2 What RASCAL Represents: Its Acronymic Underpinnings

To meet the needs of collaborative multimedia, RASCAL provides a framework that takes a unique perspective to issues of use, design and implementation. These facets form the acronym by which the framework is known.

Reflectively. The "R" in RASCAL denotes the use of *reflection* within the model as a guiding design principle and architectural technique. In particular, reflection is used to provide an "open-ended" and "causally aware" media space. The causal link which relates the meta and base levels within a reflective system offers a natural way to achieve the necessary dependencies between different entities while providing the appropriate amount of separation. It also provides a way of abstracting the way an entity behaves as well as the types of behaviours a

media space supports. Utilizing the notion of a reflective tower, reflection offers a scalable and versatile way of abstracting details of interaction and object-specific behaviour to suitable levels of detail. Consequently, media space "policy" would map to one of the highest "meta" levels while decoding of continuous media (such as video) would be at a "base" level.

Adaptive. The first "A" in RASCAL denotes the *adaptive* nature of the framework. Because collaboration is itself a co-adaptive process, any system which supports collaboration must be flexible and mirror the variation within the process it supports. The system itself must adapt in different ways, ranging from the number of media space participants to the quality and kinds of media and interactions required; the relative performance of specific components within the system as well as the policy and intent of the specific media space could also vary over the application of the architecture. Indeed, the "policy" itself can be a self-contained reflectively adaptive system in which the "policy" functions as an intelligent learning system that adapts to usage of the media space over time, function and participants.

Synchronous. The "S" in RASCAL denotes two aspects of the framework and its intended usage: (1) that the architecture supports real-time synchronously shared environments; and (2) the provision for and importance of *synchronization* within the said environments. First, the model is intended to facilitate the coordination and synchronization of a collaborative media space by providing a versatile behavioural management methodology. Secondly, within a media space, synchronization is of special importance as part of maintaining end-user semantics for media as well as their role in user interaction. Beyond the traditional notion of timeliness (i.e. temporal synchronization), the maintenance of causality (via causal synchronization) is an important aspect in collaboration. Media-specific and interaction-dependent timeliness are issues that can vary within the system and need to be supported in a open and modifiable manner.

Coordination. The "C" in RASCAL denotes the necessity of *coordination* within collaborative media space systems. Specifically, synchronization can be regarded as a necessary but insufficient condition for media space coordination. While synchronization refers to the temporal or causal correctness of events, coordination is a more extensive notion that relates to the overall coherency of media space activities. While actions can in fact be synchronized (occurring at the right time and in the right order), if they are not coordinated relative to the overall intended behaviour and function of the system, they do not serve their intended purpose; that is, whether or not events are synchronized, if they are not coordinated relative to the semantics of the collaboration, they are meaningless. Often related to planning systems, the general principle of coordination relates to the notion of organizing efforts towards achieving a goal. In collaborative systems, for example, the goal is to collaborate to some end; therefore, the coordination of the system must relate to that end and offer some way of organizing behaviours to achieve it. If the system does not maintain such behaviours, then their semantics bear no relationship to the users' role in the collaborative process. The result being a meaningless, possibly confusing, misleading and/or counter-productive effort.

Architecture. The final "AL" in RASCAL denotes *architectural framework* (or *architecture* for short). By offering a basic architecture and methodology to supports collaboration, RASCAL offers a general approach for

building media space systems that are amenable to a wide variety of uses. The architecture presents a logical topology as a way to address these issues such that system design and implementation can support variety of management schemes (ranging from coordination strategies, synchronization paradigms and functional topologies) independently of implementational structure. In particular, it offers a network-centric approach through the a "network service" approach to behavioural management.

3.2.3 Basic Framework Elements

From the user's perspective, RASCAL supplies three basic entities to interact with: the Collaborative Session, the Collaborative Object and the Collaborative Association. Their relationship is shown in Figure 3.16.

3.2.3.1 The Collaborative Session

The *Collaborative Session (CS)* is the top-level abstraction in which users participate. As such, it provides a large-grain context in which users interact with each other and use collaborative objects to achieve a specific collaborative end. The collaborative session is a specific instance of media space usage and defines the overall purpose (i.e. goal) of the collaborative activities performed by the collaborative participants within it. While the media space can consist of any number of collaborative sessions, each session is independent of any other. The goal for each session is a behavioural specification of what that instance of the collaborative process is supposed to achieve. It thereby regulates the activities (i.e. behaviours) of the participants and the tools they use within that session.

As a top-level behavioural specification, the CS does not attempt to specify all the necessary rules and requirements of all the tasks that will be done as part of the collaboration. Rather, the collaborative session serves as a union of different collaborative associations and collaborative objects (detailed next) which detail specific activities as part of the larger collaborative process.

3.2.3.2 The Collaborative Object

The *Collaborative Object (CO)* is the basic entity of user interaction and can represent a medium, application or other technological entity in the space, such as an autonomous agent. Users at different hosts communicate by manipulating these shared objects to interact with remote collaborators. Collaborative objects correspond to the "physical" entities (i.e. software/hardware) required to realize some computation. Example objects include a real-time video camera applet, a shared text editor, a synthetic MPEG movie, a remote telepointer (i.e. cursor representing a remote user's position within the shared environment) and an intelligent agent (possibly represented by a synthetically-generated avatar).

Collaborative objects, therefore, constitute the *functionality* offered to users within the collaborative process; i.e. they represent user-level collaborative "tools". Based on RASCAL's separation of concerns, each collaborative object can then be seen to have a *functional component*, which performs the domain computation, and a *behavioural component*, which controls how that functionality is exercised (Figure 3.15).

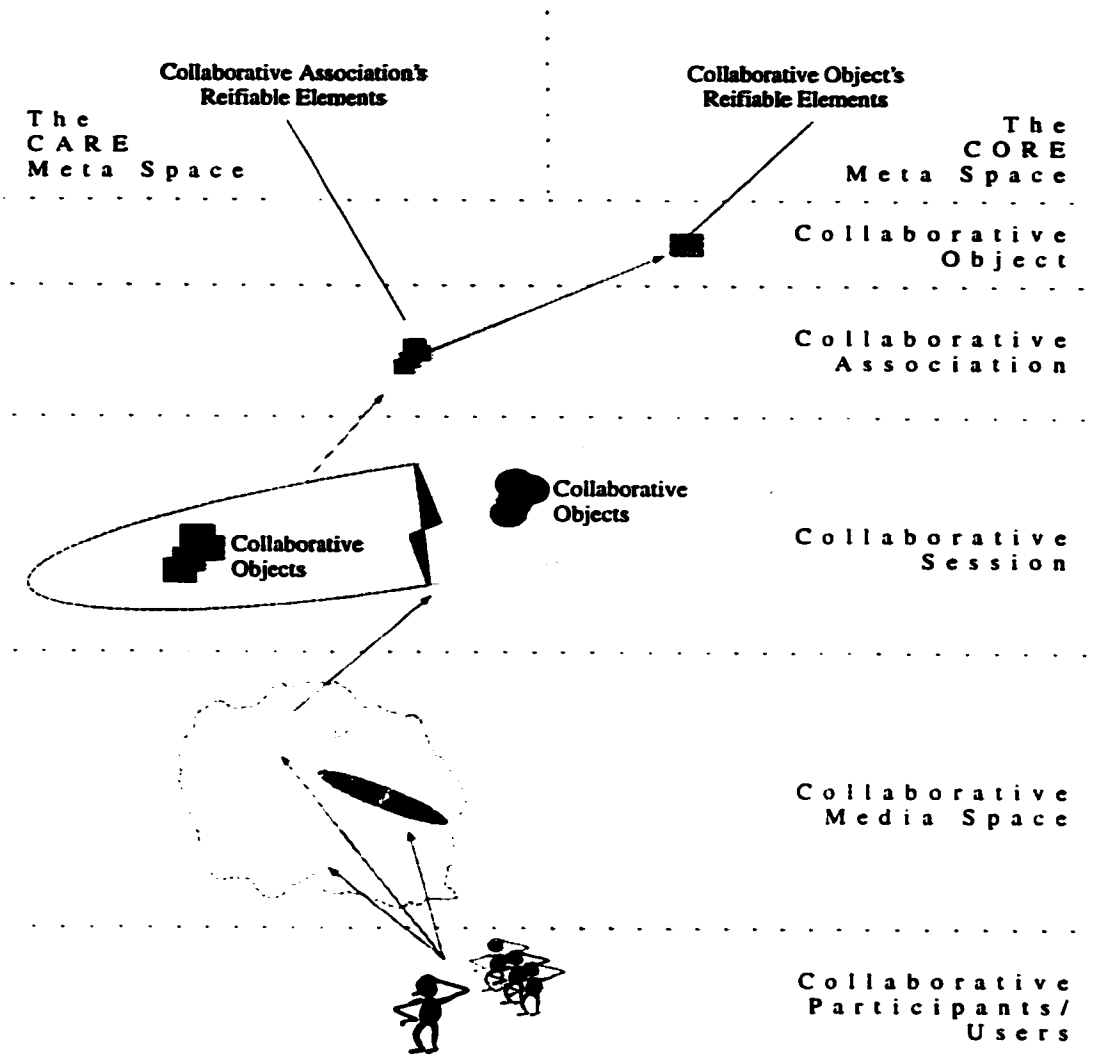


Figure 3.16: RASCAL Framework Overview

The nature of a collaborative object is that of a dynamically instantiated "functional" entity which is "inserted" for use into the collaborative media space within the bounds of a collaborative session and transparently distributed throughout the media space (Figure 3.17). Each object, therefore, has its own functional characteristics, such as different synchronization and QoS requirements. A movie object's playback, like that of a real-time audio/video conference object, must be paced appropriately, but the real-time video may require more precision and/or greater video quality. These aspects (or attributes) are inherently meta to the actual content of the object, just as behaviour was previously shown to be meta to functionality. Therefore, these attributes are represented as *meta-objects* to isolate their design and implementation details. Detailed shortly, typical meta-objects can include those responsible for synchronization, coordination, QoS, UI, functional implementation and collaborative semantics. For each object, these aspects are known as *Collaborative Object Reified Elements* and their meta-objects collectively constitute the *CORE Meta Space*.

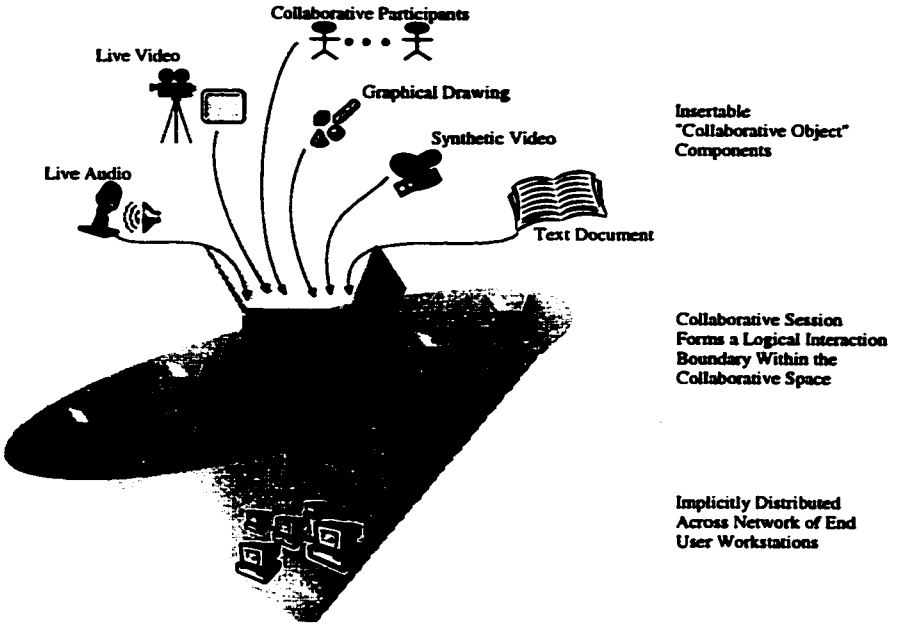


Figure 3.17: Inserting Objects into the Collaborative Space

3.2.3.3 The Collaborative Association

A *Collaborative Association (CA)* is defined as a group of collaborative objects which are currently associating (i.e. interacting) with each other. Therefore, a CA can be regarded as an *interaction group* of collaborative objects governed by some relationship. This relationship is behavioural in nature and is defined by a collaborative behavioural pattern which regulates the activities amongst the specific collaborative objects.

The main role of the CA is to provide a dynamically switchable interaction and semantic context for an object within a singular and large-grained collaborative session. Therefore, a CA effectively defines the semantics of a collaborative relationship amongst a given group of objects. Each CO can only belong to and is only semantically valid within one particular collaborative association at any time. The membership of objects within an association, however, is dynamic and can change. That is, the object may "move between" associations throughout its lifetime but the object can only interact with one group of objects at any point in time. This gives the object context in which to operate while also allowing the object to change or modify the "rules" governing its interaction with other objects by simply moving between collaborative associations. This "behavioural migration" allows an object to change with who, what and how it is interacting.

This notion of the "current collaborative association" is called an object's *Collaborative Focus (CF)*, and in an anthropomorphic sense, mimics the ability of people to change between tasks as they perform different duties and interact with different people to do their work. For example, over the duration of a business meeting, participants may "break apart" into smaller subgroups to brain-storm. In such a case, different participants may move between different subgroups (i.e. brainstorm with different people) throughout the meeting while always being part of the meeting itself. Therefore, the business meeting is an example of a collaborative session while

the different brainstorming instances are examples of collaborative associations, whose existence and composition vary over the duration of the collaboration. The CAs define the interaction's "rules and regulations" for a given set of objects which are "working together" in a collaborative context that is part of some (larger) collaborative goal.

Such changes can also be considered akin to processor migration within a distributed operating system: as a process executes only on a single processor at a given time, so can a user actively participate in a single CA at a given time; however, like a process can migrate between processors, so can a user shift collaborative focus and move between different collaborative interactions (as defined by his/her CAs). Like the logic that determines which processes should migrate is separate from the migration itself, so is the logic that controls movement between CAs separate from the CAs themselves; that is, collaborative focus determination is performed at a meta to the CA behaviour itself. This control logic can of course vary in user-level visibility, ranging from completely transparent (completely internal to the software) to directly user-controlled. This issue will be discussed further later in this chapter.

Like a collaborative object, a collaborative association is a dynamically instantiated entity that has characteristics that identify its own particular nature. Unlike a collaborative object, however, all CAs function as a means of "behavioural grouping"; the difference between them lies in their attributes which characterize that particular grouping "style". Again, these attributes are considered meta to the actual grouping function and are represented as meta-objects. As an example, an association may have specify different synchronization policies (strict, loosely coupled), particular floor control regimes, permission structures for manipulation of objects, and so forth. For each association, these aspects are known as *Collaborative Association Reified Elements* and their meta-objects collectively constitute the *CARE Meta Space*

A collaborative session can contain as many task-specific collaborative associations as desired. By default, there are two required associations:

- CA_0 *The Null CA:* default behavioural context for objects not assigned to task-specific CAs
- CA_1 *The Scenario CA:* the scenario's top-level collaborative goal and behavioural context

Consequently, in the simplest case, a collaborative session will consist of the Scenario CA to define the purpose of the collaboration and the Null CA. This would be the case in which all objects are either doing nothing (i.e. members of the Null CA) or part of the top-level Scenario CA. This could be the case given a simple scenario definition. In the general case, however, the Scenario CA and the Null CA are combined with a set of one or more task-specific "well-known" CAs that define specific behavioural patterns that manage a specific part of the collaborative effort.

3.2.4 Connecting the Elements: A Reflective Specification

To meet the needs of collaborative dynamism, the RASCAL framework was designed to facilitate a high level of adaptivity and extensibility. The intent is that a media space system based on RASCAL will be able

transform itself during its lifetime to meet the evolving needs of its users. To achieve this, the media space utilizes reflection through the use of reflective components and the appropriate meta technologies.

For a media space to be reflective, it must be able to examine, compute about and modify itself. More precisely, media space entities which are to be modifiable must be subject to reflective computation. Therefore, the modifiable aspects of such an entity must have an explicit self-representation so it can be examined and changed by a computational system. These elements are said to be reifiable and are represented as meta-objects within their entity's meta-space. Each meta-object is the actual realization of an entity's aspect in a separate yet connected manner. Therefore, the given meta-space is a logical grouping of the aspects that an entity wishes to reify and each meta-object is the self-representation of that particular aspect.

3.2.4.1 Organizational Overview

The basic framework elements introduced in the previous section form the logical core of the RASCAL approach to building collaborative media space systems. Obviously, the concept of collaborative objects and collaborative sessions is reasonably well-known; however, the novel use of an explicit representation for associations between entities and the "behind the scenes" organization of the framework elements is what provides for the novel reflective nature of the framework and its capabilities.

Summarized in Figure 3.18, the three primary RASCAL entities constitute a RASCAL collaborative media space through a simple set of compositional relationships. Simply put, an instance of a collaborative media space is a collection of collaborative sessions (being transacted over a given physical architecture as discussed later). Each session is a collection of collaborative associations, including the two default associations presented previously and all the task-specific associations that are part of the scenario in question. Each association is a set of collaborative objects that are the entities managed by the rules of the association. In addition to this object set, each association has a meta space, called the CARE Meta Space, that contains all of the elements of the association that are subject to reflective computation. Similarly, each collaborative object contains two parts: its base functionality (which may or may not be reified) and its own meta space, called the CORE Meta Space) that contains the elements of the object that are subject to reflective computation. The specifics of these meta spaces will be discussed shortly.

Based on RASCAL's intent, it is obvious that realization of the framework requires support for a high degree of parallelism. That is, because collaboration itself is a mix of different activities occurring at the same time, a system which supports it must be able to address this multiplicity. To address this need, an important aspect of the RASCAL approach is the use of *active objects* to realize various system elements. Each element, be it a software construct or a "physical" part of the media space itself, is considered an active object with its own thread of execution; each object performs computations and execute actions to fulfill its particular role within the system and the particular collaborative session.

Roles are achieved by using the appropriate active object which "knows how" to perform a particular type of activity. Enabling this type of "knowledgeable functionality" is done through the use of executable models

$$collaborativeMediaSpace = \bigcup collaborativeSession_s$$

$$collaborativeSession_s = \begin{cases} collaborativeAssociation_{Null} \\ collaborativeAssociation_s \\ \bigcup collaborativeAssociation_a \end{cases}$$

$$collaborativeAssociation_a = \begin{cases} CAREMetaSpace_a \\ \{collaborativeObject_o\} \end{cases}$$

$$collaborativeObject_o = \begin{cases} COREMetaSpace_o \\ BaseObjectFunctionality_o \end{cases}$$

$$COREMetaSpace_o = \bigcup reifiableelements_{collaborativeObject_o}$$

$$CAREMetaSpace_a = \bigcup reifiableelements_{collaborativeAssociation_a}$$

Figure 3.18: Summary of RASCAL Framework Relationships

with the inherent ability to participate in reflective acts. The content of each model provides the logic of the particular role and the use of reflection allows the appropriate level of abstraction and separation of concerns between different objects. Effectively, the use of "reflective models" allows objects to actively and independently manage themselves and because the model is executable, modifications to the flow of execution is achieved by modifying the (design) model itself. Consequently, consistency is more easily maintained and modifications to an executing entity is the same as modifying its model.

Therefore, entities within the space are derived from a meta type called *REXS* which stands for *Reflective EXecution Specification*. REXS is a means to specify the use an active object (i.e. executable) paradigm with "native" support for reflection. The actual realization of REXS in terms of implementation can vary and will be discussed further in upcoming sections, including that on RASCAL's class structure.

3.2.4.2 Using Meta Technologies: Meta-Objects to Meta Spaces

Use of the meta-level approach has been shown in several places throughout this work. Combined with reflection, the meta-level hierarchy has been shown as key to separating areas of concerns in a structured, abstract and useful manner. These separated elements have been mentioned previously using the notion of a meta-object which results in the formation of a meta space. However, what meta-objects and meta spaces are as well as how they are related within RASCAL have not yet been addressed in detail.

In general, a meta-object is simply an object, except that it is considered at a meta level relative to some object which functions as its base. By virtue of definition, these objects represent some "higher" level of description or detail about the base object. The result is that (significant) aspects of an entity are typically represented using their own distinct object at meta level relative to the given base entity. It is the collection of such meta-objects that form the meta space for a given entity. The results include increased reusability, extensibility and understandability via clearer and more straightforward organization of responsibility within the system.

The relationship between a meta-object and its base is done using a *meta-object protocol* (or *MOP*) which allows inspection and intercession between the different levels. However, given that any one level can be regarded as base to the one above it, objects at the meta-level are also open to reflection. Similarly, objects at the meta-meta-level are again open to reflection and have an associated meta-meta-meta-space. This recursive process continues providing an infinite reflective tower. However, while such a structure exists in theory, a given level is only instantiated when required, giving a finite representation of an infinite structure. Therefore, while the definition of the MOP can be extremely general, it is typically related to the domain of the objects in question. Consequently, a set of smaller meta-object protocols could be used within a system that has a diverse meta space.

Within RASCAL, meta-level characteristics are organized relative to the three main collaborative entities:

1. Collaborative Object: The CORE Meta Space
2. Collaborative Association: The CARE Meta Space
3. Collaborative Session: The COSE Meta Space

Briefly, aspects specific to a collaborative object are addressed by meta-objects contained within its CORE meta space while inter-object and group issues are dealt with in the CARE and COSE meta spaces. The specifics of each meta space, along with considerations for meta-object protocol design, are detailed next.

3.2.4.2.1 The Collaborative Object's Meta Space

The CORE Meta Space contains all the aspects of a given collaborative object that the system wishes to make available for dynamic run-time modification via reflection.

Shown in Figure 3.19, these meta-objects provide support for common object attributes such as its user interface, functional implementation, behavioural definition (including synchronization), QoS, security and collaborative semantics. The list of attributes that are reified, however, is open-ended and is ultimately at the discretion of the system designer. However, these particular aspects offer reasonable flexibility for a collaborative object as follows:

Functional Implementation and Behavioural Definition. To the user, the essence of a software tool is what it does (i.e. its functionality) and how it does it (i.e. its behaviour). As a collaborative object is a logical entity within the media space, it logically has function within that space as well as a behaviour. However, by reifying

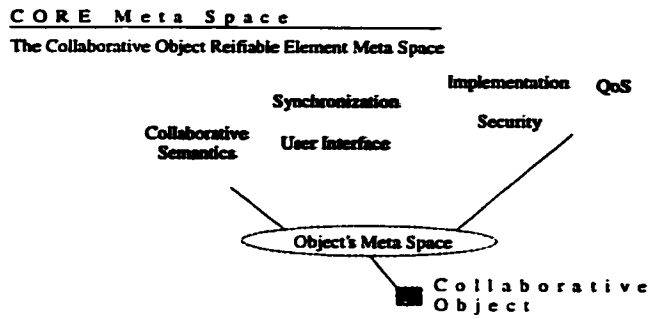


Figure 3.19: Object Meta Space

both of these facets separately, the domain-specific functional code and its behavioural definition can be considered separately. This includes any synchronization specifics that are to be associated with the object, including mechanism, paradigm selection, asynchrony tolerance and so forth.

As shown in Figure 3.20, the collaborative object always reifies its behaviour to a separate meta-object so that all the complex issues of behavioural management are not embedded within the functional logic. Ideally, the functional implementation of the tool would also be reified and addressed separately. However, this separation is not completely clear when attempting to use legacy components. Therefore, the enforcement of an externally specified behaviour upon the legacy component (i.e. the collaborative object) may in fact be the specification of how to use the inherent legacy behaviour and functionality relative to the new external behaviour. This issue will be addressed further in the discussion of the conceptual prototype.

User Interface. Traditionally, the user interface of a software tool has been a fixed part of the software itself. Recently, however, trends towards UI customization have offered users the ability to modify the way they interact with a given program. Therefore, by taking the user interface of a software tool as a meta-level concern relative to its functionality, by reifying the UI, different visuals and/or interaction mechanisms could be employed for a given functional base depending on hardware capability. By considering the UI separately from functionality, different users could potentially utilize different UI technologies within the same collaborative effort. For example: manipulation of a shared robotic arm could be achieved as an on-screen graphic manipulated by a mouse or through the use of an actual physical robotic arm assembly with force-feedback. While the functional logic must interact with the events/responses to/from the UI, it should not be concerned

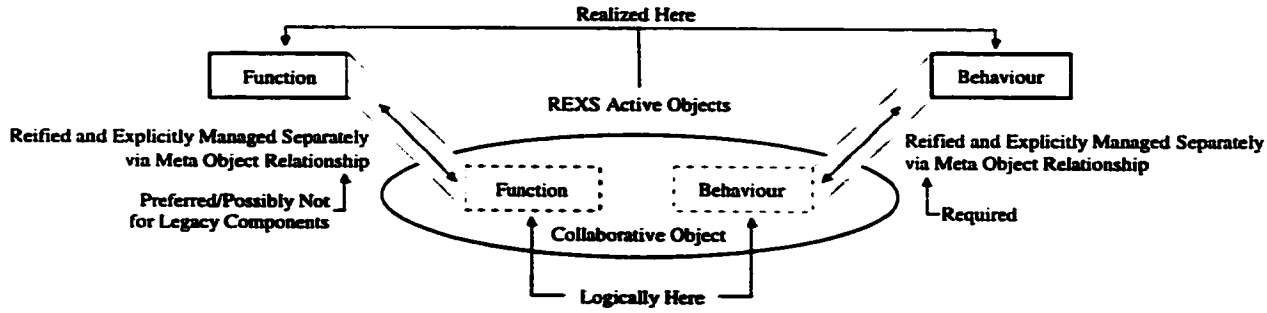


Figure 3.20: A Collaborative Object's Behavioural and Functional Meta-Objects

with the actual physicality of the interface itself. This idea will be discussed further in the context of the conceptual prototype in subsequent chapters.

Security. Much like permissions and ownership associated with multi-user file systems, security considerations could be relative to a collaborative object. Examples include limiting its use to certain users or types of operations. Through separate consideration, however, evolving security needs could be more easily introduced than if specific usage restrictions were embedded within the functional code itself.

Quality of Service/QoS. The ability to introduce new and adapt existing quality assurance measures is made easier and less functionally dependant by isolating the specification and mechanisms that deal with the QoS of individual objects. The traditional interpretation of QoS, as applied to networking issues, is usually dealt with separately from other functionality. Because QoS issues are often related to environment influences (such as network bandwidth, processing overhead and so forth), their separate consideration removes the complexity associated with monitoring and adjusting these aspects independent of the actual (and possible invariant) function of the object. Variance in object function due to QoS issues, of course, can be dealt with by the functional implementation being able to select execution options based on information from the QoS module.

Collaborative Semantics. The utility, applicability and meaning of an object in a given context in part depends not only on the context but the capabilities of the object. The notion of "collaborative semantics" allows an object to specify restrictions or other useful properties about itself that can be queried by users or other parts of the system (such as a collaborative association, for example) to determine if certain actions are appropriately applied to that object. This could range from the object being able to support single or multiple users, whether object versioning (for different synchronization paradigms and user interfaces) are applicable, and if so, what the resolution mechanism and/or policy are.

3.2.4.2.2 The Collaborative Association's Meta Space

Unlike many systems in which entity relationships are inherent (either via system design or code dependencies), RASCAL explicitly represents inter-object relationships as independently realizable and computational entities – i.e. it reifies them. Called a "collaborative association" with the RASCAL nomenclature, a collaborative association between objects can therefore be examined, computed about and adapted according to the logic which defines that particular relationship.

As a reifiable entity, each collaborative association has a meta space associated with it that defines the aspects of that CA that are open to dynamic run-time modification via reflection. Called the *CARE Meta Space*, the CA's meta space contains the meta-objects that the association uses to describe itself, however simple/complex, open/closed it is designed to be. An example CARE Meta Space is shown in Figure 3.21 and while the list of attributes that can be reified is at the discretion of the system designer, certain aspects are key to providing the intended flexibility of the RASCAL framework:

Behavioural Definition. The function of a collaborative association is to group interacting entities together according to some behavioural logic. Like the behaviour of a collaborative object, this logic is not an implicit

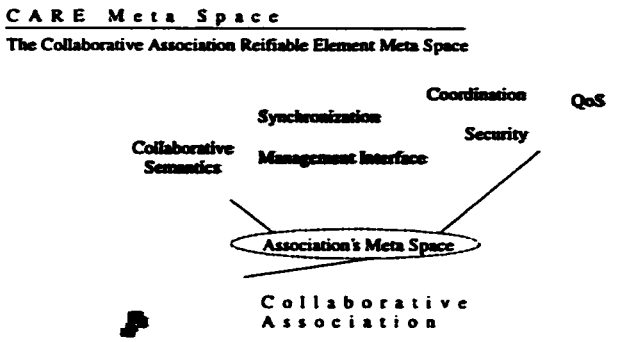


Figure 3.21: Association Meta Space

part of the association per se but provided via a meta-object as part of its meta space. Through this explicit realization of the association and its logic, both can be changed at run-time thereby not only allowing the system designer but the user and/or the run-time computational system to reflect on the their state and make the appropriate changes. Consequently, the synchronization and coordination logic of the relationship (including issues of ordering, priority, timeliness and accuracy) can be changed dynamically as part of modifying the association's behaviour.

Security. As with a collaborative object, security considerations can be taken relative to a collaborative association. Possibilities include limiting participation to a specific (or certain number of) users or disallowing certain types of operations. An example would be to deny "edit access" for a user across all kinds of applications rather than denying it on a per object (i.e. application) basis. Separate consideration also helps facilitates less complex association logic and evolving security needs can be more easily introduced than if usage restrictions are embedded within the association code itself.

Quality of Service/QoS. Similar to that of a collaborative object, the QoS for a collaborative association isolates issues related to the quality of the association (i.e. how well the association is maintained). How well a particular association is maintained refers to how stringent the association logic is enforced subject to user preferences, system resources and so forth. Consequently, the QoS of a collaboration defines how good the relationship is enforced, effectively forming the notion of *Quality of Interaction*, or *QOI*. This notion is explored further in the subsequent chapter. In addition, separate consideration also helps facilitates less complex association logic and evolving QoS facilities can be more easily introduced than if embedded within the association code itself.

Management Interface. While standard software tools traditionally have the notion of a user interface, the concept of a relationship between them does not. However, in as much as there is an "embodied" computational representation of the relationship between said objects, there is the need to somehow manipulate (or manage) it. Therefore, the management interface of the collaborative association is the means by which the association can be controlled. This could manifest itself as a user-centric graphical user interface or an internal system-based control system. Regardless, the principle is that there is a way to interact with the association as

a computational entity and consequently, there needs to be a way to control it. This is particularly true in relation to the association's collaborative semantics, as discussed next.

Collaborative Semantics. The utility, applicability and meaning of an association in part depends on the collaborative objects and what their pattern of interaction is supposed to be for a given context. The notion of "collaborative semantics" allows an association to specify the range of rules, restrictions or other useful properties that can be queried by association's behaviour logic to determine if certain contextual requirements are being met. A key part of this context is determining which objects are part of a dynamically defined group of interacting entities. This determination is the responsibility of the *Collaborative Association Membership Policy*, or *CAMP*.

To explain the rationale behind CAMP, consider the following scenario:

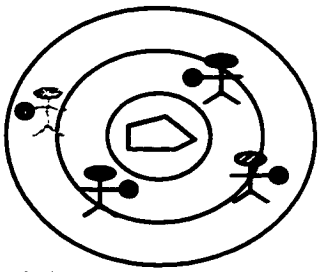
> A text document is being jointly edited by a group; however, only three people at any one time can actually edit the document while the rest observe.

For this situation, there is a need to determine which users are actually editing the document and which are observing the editing process. That is, those who actually editing the document are part of the "text edit" CA and only need to be regulated in terms of their interaction with the document and each other. The CA's membership policy (i.e. CAMP) therefore determines which entities are actually part of the behaviour being governed by the association.

Inclusion in a collaborative association is based on the notion of a *logical boundary* which delimits an entity's membership in a CA. Since this membership policy is a meta-level attribute to each CA, each association can set its own arbitrarily complex membership requirements. Common membership policies can also be defined separately and changed independently of actual CA logic; that is, who is part of the association is independent of the association itself. Subsequently, a CAMP effectively defines *semantic-based admission (and departure) control* for a given CA, since membership in a CA is to be a part of the interaction semantic it defines.

Consider for example, a collaborative environments based on 3D virtual reality which would define the CAMP boundary using physical proximity vis-à-vis spatial coordinates. Such an approach is also easily mapped to a two-dimensional graphical user interface (GUI), in which the proximity of participant telepointers (remote cursors) relative to each other and/or editable objects can be used to determine access rights (Figure 3.22).

These coordinate-based boundary tests, for example, would be most reminiscent of collaborative virtual environments in which interfaces based on avatars and 3D graphical objects use "geography" to limit object awareness and interaction. Alternatively, a specific user interface could be provided to define CAs and select object's into and out of them. Other possible rule sets could involve complex monitoring logic do grouping based on object-specific contents (i.e. voice analysis of participant teleconferencing using keywords to identify/select objects). User-level and system-based CAMP logic can be used either alone or in combination to support various options for collaborative focus determination, ranging from completely manual to completely



Proximity CAMP: Membership and
Priority Based on Distance from Object

Figure 3.22: Illustrating A Proximity CAMP

automatic, or one in which users could supplement/override basic automatic decisions (see Figure 3.23). This flexibility is part of supporting a wide range of user interface and interaction techniques.

How the system performs this detection and how it performs the migration could be arbitrarily complex and may vary depending on the objects in question (i.e. types of activity and tools being used), as well as other system-dependent conditions (i.e. resource utilization) and the specific behaviours being utilized in the groups. Therefore, there is a need to provide an adaptive (i.e. changeable) mechanism for realizing arbitrarily complex behaviours and how to modify them on the fly. Such diversity in boundary determination exemplifies the need to provide CAMP logic that can be specified separately, shared between objects and defined in a manner such that it can modify itself independently of the CA logic; for example: adding new keywords, changing proximity sensitivity, the ability to ignore or force the inclusion of certain (categories of) objects in a particular kind of association, etc.

CAMP Type		Explanation	Illustration
Proximity	Absolute Coordinates	"Physical" locales/regions within the space are used to map entities to a collaborative association based on their coordinates within the space	Space divided into rooms, offices, etc.
	Delta Coordinates	Proximity of entities within the space are used to define whether or not they are associated; i.e. the possibility of interaction based on distance between them	Avatars and/or objects close enough to "see" each other
Semantic	User Specification	Users manually group entities into a given association via some user interface	Object selection via mouse click/drag
	System Specification	The system itself groups entities into a CA based on meta-level computations; possibilities include: • *Interaction Patterns* – AI-based pattern recognition such as standard or relative movements • *Entity Type* – such as support for synthetic/live media pairing, multi-user interactive document editting • *Behavioural History* – such that a user's characteristic mannerisms are recognized (over time) and form a basic "behavioural profile" which typifies how he/she works and interacts	Computational equivalent of "habitual behaviour" and/or "personal favourites"

Figure 3.23: Possible Types of Collaborative Association Membership Policies

A collaborative association's membership policy would be represented like any other policy within RASCAL, as a collaborative behavioural pattern. Consequently, adherence to a CAMP is to execute according to its behavioural pattern. And as such patterns can address multiple behavioural characteristics, they may involve processing at more than one level of behavioural management. For example, a simple test could involve that the participant's cursor is within a certain bounding rectangle for a required length of time (each of these aspects somehow being relevantly determined). Alternatively, the test of group membership could involve both such a "proximity test" combined with a higher level's monitoring of the particular user's activity level within its former group. For example, if the overhead of coordinating people in a joint editing scenario is high, then the user join logic may wish to analyze the likelihood based on past participant in editing activities to decide the type of inclusion the user is granted with respect to this new group. For example, someone who sits back and observes the joint editing would apt not to need the rigor of synchronization that an active editor in the scenario would require in order to stay in sync with his/her fellow editors. Such an example utilizes the notion of a behavioural history as a meta characteristic of a participant within a space. Another such characteristic could be the security concerns of both the scenario and/or participant in question.

3.2.4.2.3 The Collaborative Session's Meta Space

As introduced previously, the collaborative session effectively functions as a logical and existential boundary for user interaction within a media space.. In terms of collaborative activity, however, it is defined as a union of existing collaborative associations and therefore its capabilities are those of its constituent CAs. The session's meta space can therefore be seen as a CARE union (Figure 3.24) and is simply known (for completeness) as the COllaborative SEssion Meta Space, or COSE Meta Space.

3.2.5 Methodological Outline: Building a Collaborative Media Space with RASCAL

This section outlines the concrete aspects of the framework and considerations for real-world realization of a media space based on RASCAL. It presents the various facets of building a system ranging from techniques for

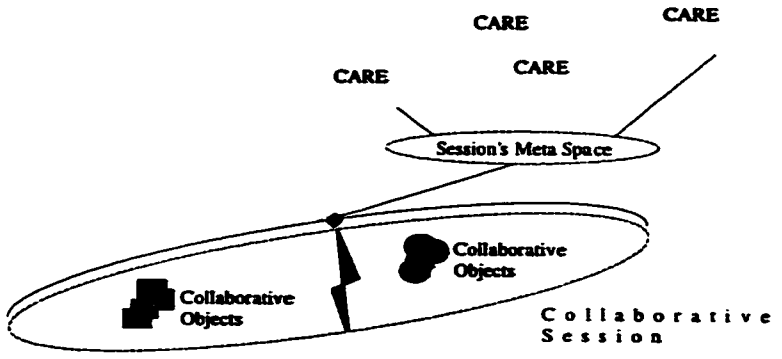


Figure 3.24: Session Meta Space

addressing entity configuration to the basic class structure and the underlying communication topology. Example systems and various application scenarios are discussed further in Chapter Four.

3.2.5.1 Configuring the Conceptual: Formulating the Collaborative Design

As part of the RASCAL methodology, three diagrams can be used to describe the behavioural organization of entities relative to each other and their semantic/working relationships. Due to the dynamic nature of entity composition, each diagram effectively serves as a "starting point" or "snapshot" of entity organization at a given point in time. Additionally, while each diagram is intended to aid in a specific aspect of collaborative media space design, no diagram is intended to dictate how specific entities are to be implemented in concrete terms. The various diagrams and their individual formats are used together to provide different perspectives of the collaborative space and its entities instead of using a single convoluted notation. Correspondence between them is aided through the use of numeric labels and specific shapes as outlined in Table 3.2. The diagrams are:

1. The Conceptual Scenario Illustration Grid — CSIG
2. The CA Allocation Diagram — CAAD
3. The Collaborative Behavioural Tree — CBT
4. The CA Pattern Reference — CAPRE

The *Conceptual Scenario Illustration Grid* (or CSIG) is used to illustrate the organization of the collaborative scenario as a conceptual grid, showing entities and their groupings based on how the designer envisions their interaction at a given point in the collaboration. This grid then forms the basis of the *CA Allocation Diagram* (Figure 3.25), which is the primary user-level means of illustrating how different entities are grouped into various collaborative associations. CAAD shows the behavioural hierarchy via nesting grid elements within each other while semantically disjoint entities are shown as physically separate elements (i.e. in separate grid partitions). Entities that migrate between different CAs as part of their standard definition can be shown in their own grid (or within their parent if convenient) with arrows showing their possible migration paths (if specific or well-known in the given instance). As a 'fish-eye' view of the behavioural organization of a collaborative session, this diagram has the potential to form the basis of a behavioural management GUI for both individual collaborative objects as well as their associations.

Entity		Notation
Collaborative Session		
Collaborative Association	Composite	
	Singleton	

Table 3.2: Cross-Diagram Symbol Notation

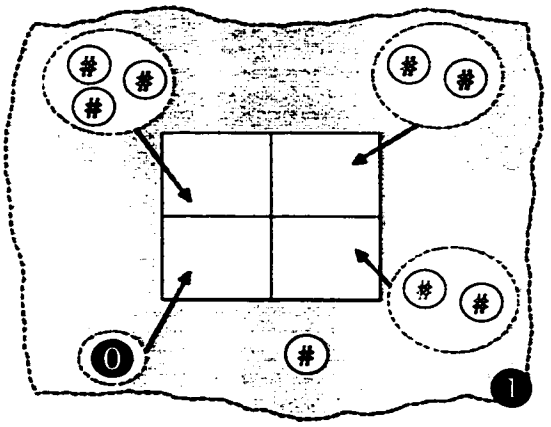


Figure 3.25: Example CAAD — the CA Allocation Diagram

The collaborative associations shown in CAAD therefore offer a spatially-oriented view of interactions between (groups of) collaborative objects. However, in terms of recursively complex behaviour patterns, different collaborative groupings with different semantics can be seen to form constituent behavioural subgraphs within a *Collaborative Behavioural Tree* (see Figure 3.26). Within the CBT, each of these collaborative groupings exists as a node along a *semantically associative branch* which can be coordinated independently according to relevant characteristics and according to a higher-level behavioural pattern that provides its "larger" collaborative context. Ultimately, the top-level root behavioural pattern defines the goal of the entire collaboration itself which is the top of the reflective behavioural management hierarchy (i.e. reflective tower) for the collaborative effort as a whole. The CBT is therefore a logical graph-based notation used to specify the hierarchical organization of RASCAL entities within a specific instance of a collaborative session. The general form of a CBT has two requisite nodes: the root (formed by the Scenario CA) and its child (the Null CA). All other collaborative association nodes are optional and can be dynamically added, removed or re-connected within the tree relative to their semantic association with other entities in the collaboration.

Table 3.3 summarizes the notation and composition of a CBT in which the optional nodes are either *semantic composites* or *semantic singletons*. While each of these entities is a collaborative association in itself, they differ in whether they exist as a stand-alone behavioural component or one that functions as an organization of more primitive entities. Self-contained components that form the lowest-level of the behavioural hierarchy as

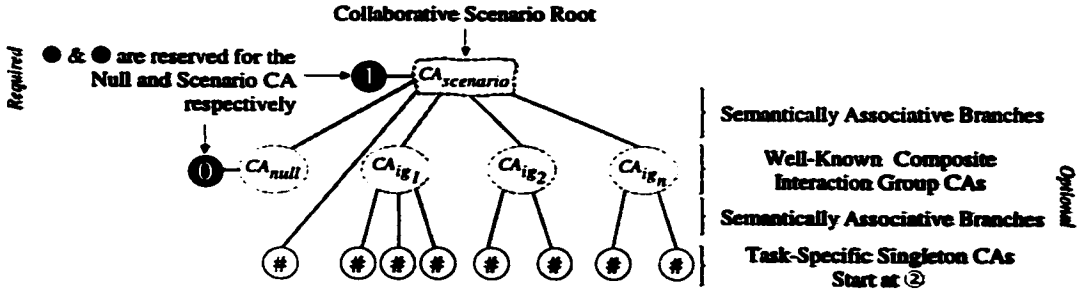


Figure 3.26: General Form of a CBT — the Collaborative Behaviour Tree

Tree		Composition			
Node	Notation	Semantic	Format	Parent/Child Role	
Root		"Semantic composites" whose behaviour is defined to make sense as a coordinated hierarchy of more primitive patterns	Behavioural Pattern	Leaf & Non-Leaf Children	Child nodes are more primitive or task-specific Collaborative Associations
Non-Leaf					
Leaf		"Semantic singletons" whose behaviour is defined to make sense as a single entity	Behavioural Component	No Children	Leaves are low-level, self-contained hierarchies containing semantically integral policy and mechanism

Table 3.3: Summary of CBT Notation and Composition

part of the collaborative object (see upcoming Figure 3.31) are the leaves of the CBT while intermediary non-leaf nodes represent more abstract hierarchical behavioural patterns.

As the collaboration evolves, and differing behaviours and behavioural groupings are formed, the topology of the tree and its constituent nodes dynamically changes. Therefore, the CBT can be taken as a real-time representation of the behavioural makeup of the collaborative session. The use of reflection and the meta-level hierarchy maps to different levels within the tree and movement between base and meta levels maps to movement between nodes. The semantically associative branches of the CBT, however, represent behavioural connectivity within the collaboration; therefore, the integration of additional meta-objects and their corresponding meta spaces (such as shown in Figure 3.27), results in a *multi-dimensional collaborative organization tree*. In such a tree, the extra dimensions are formed from non-behavioural edges which relate CBT nodes to the various meta-objects described earlier in this chapter. In each case, however, by following the edges, entities can perform reflective acts on those aspects which are connected to it.

The CA Pattern Reference (shown in Figure 3.28) is used to supplement the CBT in that instead of hilighting abstract behavioural composition, CAPRE details individual low-level patterns. Specifically, the CA Pattern

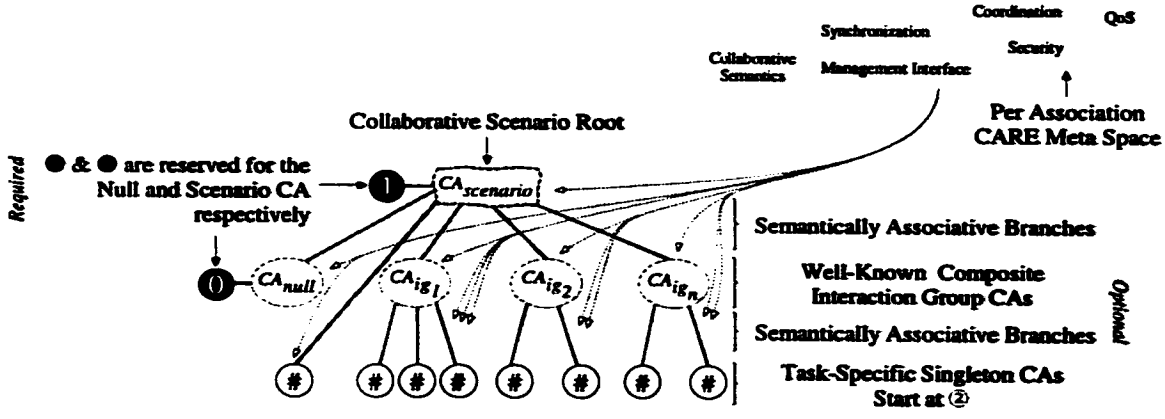


Figure 3.27: An Augmented CBT With Related CARE Meta Spaces

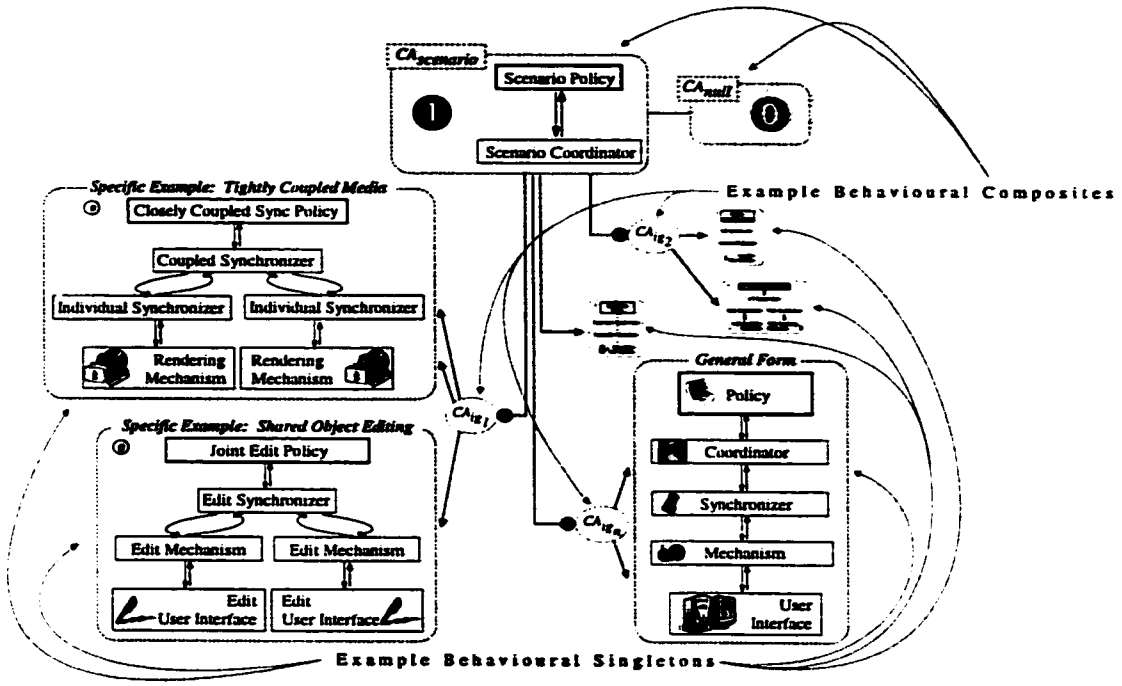


Figure 3.28: Example CAPRE — the CA Pattern Reference

Reference illustrates the semantic singletons that constitute the leaves of the CBT. These patterns illustrate how the system's collaborative functionality is "attached" to its behaviour at the lowest end of the behavioural hierarchy's reflective tower. Variations of CAPRE can be used to show different levels of detail and/or design preferences. Figure 3.29, for example, illustrates the general semantic singleton CAPRE and its integration of the CORE and CARE meta spaces for a generic CBT leaf. Figure 3.30, on the other hand, illustrates two design variants of a CBT leaf for specific usage contexts (editing vs. conversation). In each case, the meta-objects of concern are italicized within the meta space annotation.

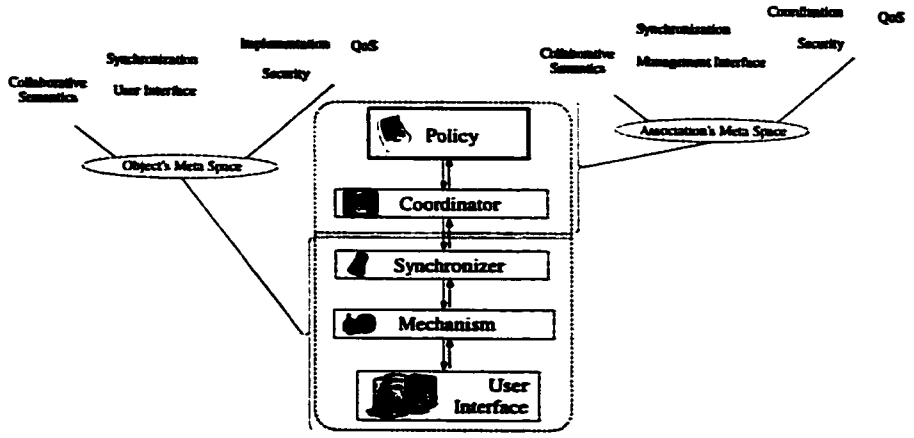


Figure 3.29: General CBT Leaf With CARE/CORE Meta Spaces

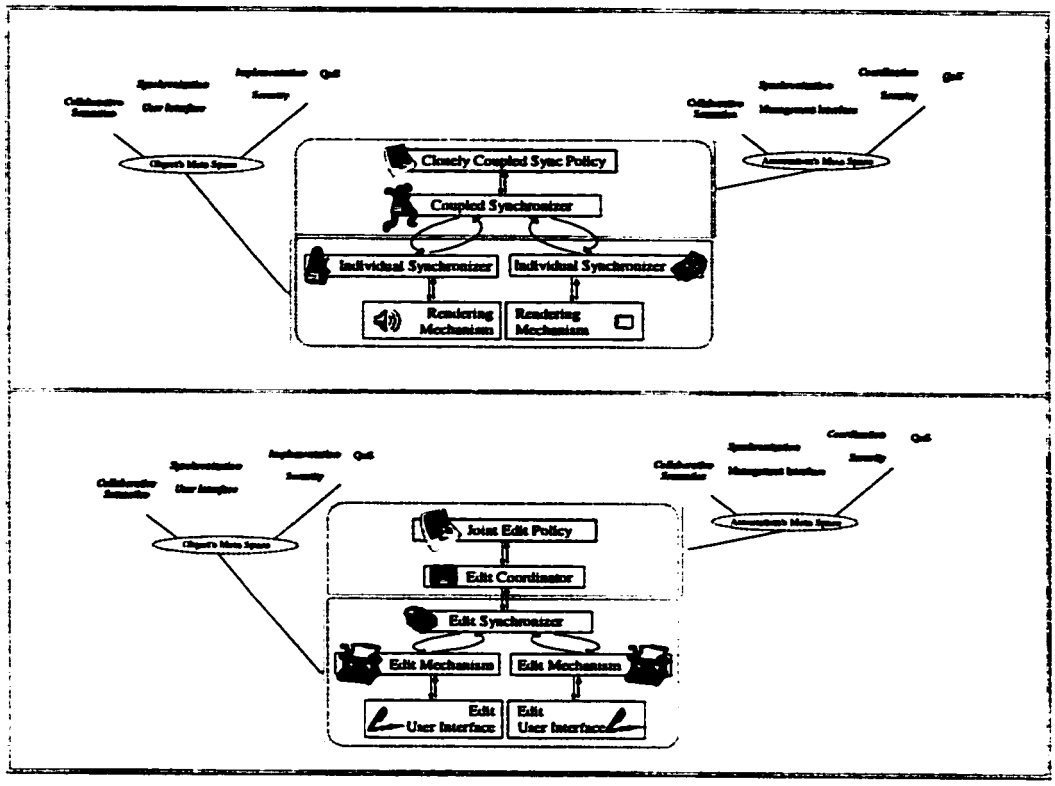


Figure 3.30: Specific CA Leaves With CARE/CORE Meta Spaces

3.2.5.2 Concretizing the Conceptual: The RASCAL Class Structure

In support of RASCAL's conceptual framework and methodology, a concrete class structure is now presented. Shown in Figure 3.31, an annotated UML [FK97] specification is used to show the classes, their relationships and how they relate to the various levels of abstraction within the collaborative paradigm. These specification levels, their scope and example entities are outlined in Table 3.4.

The various collaborative entities within RASCAL media space exist at different levels of abstraction, with differing functional purposes and behavioural characteristics. These differences are mirrored in the class framework with their correspondence listed in Table 3.5 and shown by means of example in Figure 3.33.

RASCAL's basic tenet of separating coordination from execution is addressed via composition within the *Object* class via the *Object Functionality* and *Object Behaviour* subclasses. This ideal division is intended for custom RASCAL systems; however, those using legacy components may not have such a clear separation in terms of implementation. In such a case, the composite *Object Behaviour* effectively becomes a wrapper around the existing object with its intrinsic behaviour. This issue is discussed further in subsequent chapters.

While created within the bounds of a collaborative session, collaborative objects are utilized within the bounds of a collaborative association. In this sense, they are syntactically global while semantically local to their current CA. However, since a session is actually a union of CAs, an object's membership in either the session

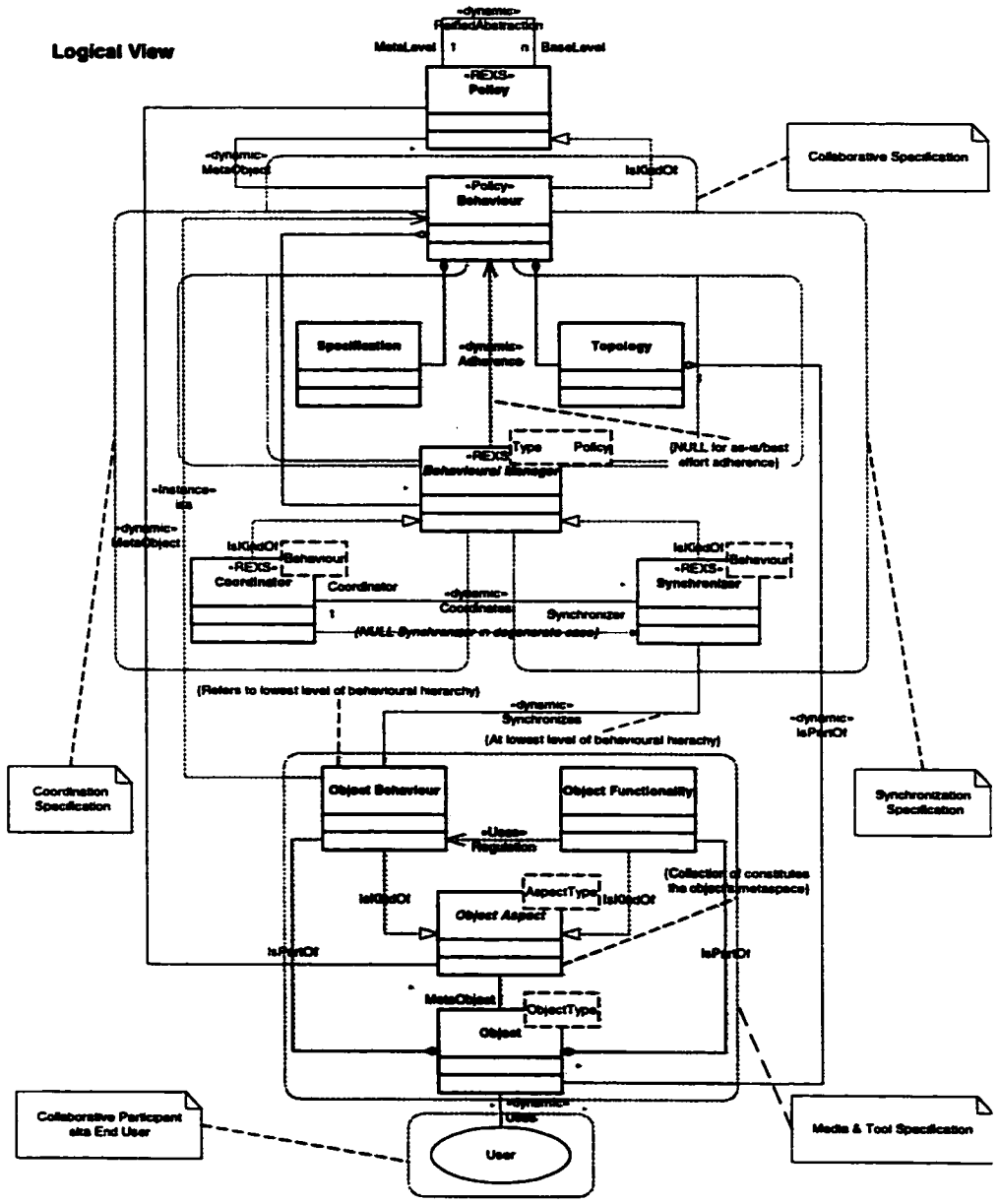


Figure 3.31: RASCAL Class Framework in Annotated UML Notation

or any particular CA can be represented as an aggregation of the *Object* class relative to the *Behaviour* (via its *Topology* subclass – see Figure 3.31).

While the concept/class mapping of the collaborative object is reasonably straight–forward, that for a collaborative association is less so. As stated earlier, the intent of each CA is to be an active behavioural entity facilitated through the use of executable models. Within Figure 3.31, the use of executable models combined with the requirement to participate in reflective acts is represented through the use of a *stereotype* [FK97] called *REXS*. Short for *Reflective EXecution Specification*, REXS specifies the use an active object (i.e. executable) paradigm with "native" support for reflection. The actual realization of REXS is implementation dependent and

Specification Level	Scope	Example
Collaborative	High-level management of goal-oriented behaviours	Business Meeting
Coordination	Mid-level management of group-oriented (sub-goal) behaviours	Brainstorming Group
Synchronization	Low-level management of specific object(s) behaviour	Audio/Video Lip Sync
Media/Tool	Object-specific realization (logical location of behaviour and function)	Teleconferencing Applet

Table 3.4: Specification Levels and Their Scope

various possible executable models exist, such as those based on petrinets and finite state machines. The major difficulty with these techniques is to facilitate their reflectivity. This issue will be discussed further in subsequent chapters.

REXS forms the foundation for all RASCAL entities. As the basis for the *Policy* class, it enables a RASCAL *Policy* to actively execute and compute about itself in a reflective manner. The Policy, in turn, is the basis from which all collaborative entities are derived. For example, the *Behaviour* class facilitates creation of behavioural components, patterns and hierarchies through its ability to reify itself and form a reflective behavioural tower. This is enabled via the *Policy* class in which meta-level policies constitute meta-level behaviours. Figure 3.32 illustrates REXS relative to a behavioural pattern built as a reflective tower of active objects. In such a definition, the bottom level of the Behaviour hierarchy interacts with collaborative objects (instances of the class *Object*) via a *Behavioural Manager(s)* (realized as a *Coordinator* or *Synchronizer*). The complexity of the behavioural definition is addressed by allowing semantic levels to be mapped to different Behaviours which are reflectively linked as meta-behaviours (i.e. a *Policy*).

The collaborative association is therefore represented using the *Behaviour* class. Each CA, as a behavioural pattern, is an instance of a reflective behavioural tower defined via REXS and the *Behaviour* and *Policy* classes. At each level of the tower, the Behaviour's *Topology* refers to its constituent base members (i.e. objects or sub-behaviours). The *Specification* refers to the logic that instance of the *Behaviour* class (i.e. that level of the behavioural tower, or part of the behavioural pattern) implements.

This approach promotes scalability and allows the complexity of the association to be distributed amongst the different levels of the tower. The result, in which CAs offer increasing levels of complexity (or "detail") as the

Framework Entity	Primary Class	Supporting Classes	Specification Level
Collaborative Object	Object (Functionality & Behaviour)	Policy	Media & Tool
Collaborative Association	Behaviour (Specification & Topology)	Behaviour Manager (Coordinator & Synchronizer), Policy	Coordination & Synchronization
Collaborative Session	Behaviour (Specification & Topology)	Behaviour Manager (Coordinator, Synchronizer), Policy	Collaborative

Table 3.5: Framework Entities, Classes and Specification Levels

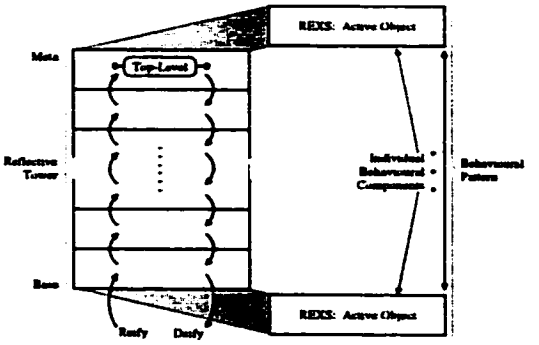


Figure 3.32: Behavioural Pattern as Reflective Tower of Active Objects

tower progresses, is known as *Progressive Associativity*. Akin to "progressive resolution encoding" used for JPEG images, in which differing levels of image detail exist within the same encoding, the CA's quality of interaction (QoI) can be adjusted simply by traversing a different number of levels in the behavioural tower.

Use of the various classes over the lifecycle of a RASCAL media space is outlined Table 3.6. As shown in Figure 3.33, a media space exists via the realization of collaborative sessions over the RASCAL communications architecture (as detailed shortly). While generally self-explanatory, the lifecycle consists of 4 basic phases: Creation, Restoration, Application and Termination.

The *Creation* phase deals with the initial realization of a collaborative session. As explained earlier, creating an session maps to creating the default null and scenario CAs. The *Restoration* phase provides for dealing with

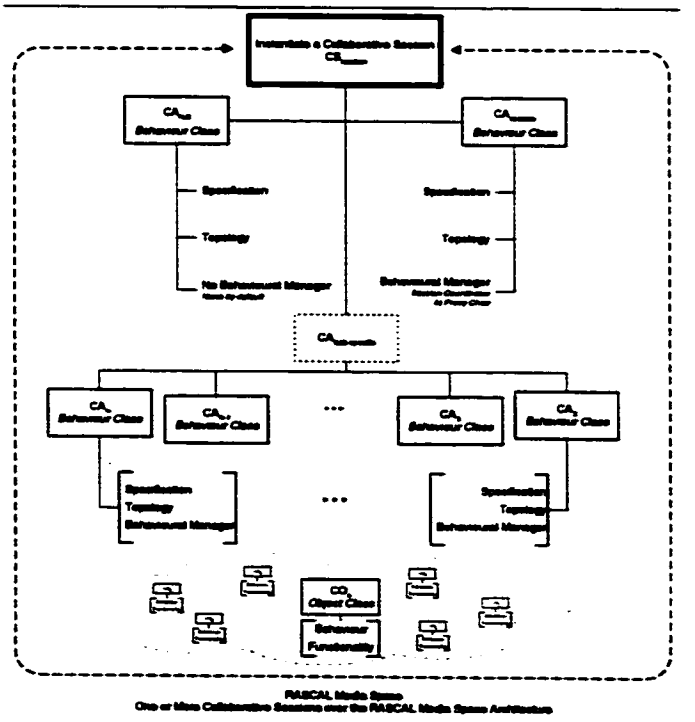


Figure 3.33: Example Media Space Instance

#	Phase & Involved Steps		Result	Classes
1	Creation	Instantiate collaborative session $CS_{scenario}$	Creates "empty" session $CS_{scenario}$ with base associations: CA_{null} $CA_{scenario}$	Behaviour
2.1	Restoration: persistence across session invocations	Create any pre-defined collaborative objects that are part of the scenario	Create media/tool objects CO_o and populate $CA_{scenario}$ with them	Object
2.2		Create any pre-defined collaborative associations that are part of the scenario	Create CA_a where $a \geq 2$	Behaviour
2.3		Migrate collaborative objects from $CA_{scenario}$ to the appropriate $CA_{task\text{-}specific}$ as required	Move CO_o from scenario's CA to its appropriate CA:	Object/Behaviour
			$\forall a, o \in CS_{scenario}$ $\begin{cases} CA_{scenario} = CA_{scenario} - CO_o \\ CA_a = CA_a + CO_o \text{ (as required)} \end{cases}$	
3	Application	Utilize the space and its constituent entities for purpose of collaboration	Varies; can include: Dynamic CO creation, manipulation, destruction Dynamic CA creation, manipulation, destruction	Varies
4	Termination	Collaborative session $CS_{scenario}$ ceases to exist	All COs and CAs are deleted; when session $CS_{scenario}$ is "empty", it is deleted	Object/Behaviour

Table 3.6: Media Space Lifecycle

entity persistence across session invocations (at both object and behavioural levels). For example, users may wish to "continue their meeting later". If so, they would persist (i.e. "save" to secondary storage) their session (via its collaborative object and associations) for later use. While collaborative associations are designed to transparently "go away" when empty of objects, their definition can persist. This supports continuation and re-use of behaviours while also mirroring real-life collaborative efforts which only exist when there are participants actively performing some action.

The *Application* phase is the "active" portion of media space usage; during this phase, the media space is being used by its participants. Actions and usage of the class structure within this phase will vary widely, but would typically involve those outlined in Table 3.7.

The *Termination* phase involves "tearing down" the session by deleting any existing collaborative objects and associations. When all such entities are removed, the session ceases to exist. Support for persistence would be part of this phase, with each entity being responsible for its own persistence. An collaborative object, for example, would have its persistence and termination logic handled by the tool which actually implements the object's functionality. The persistence and termination of a behaviour, on the other hand, would depend on

Entity	Action	Explanation
Object	Add	Dynamic creation of a new collaborative object within a collaborative session. It is inserted into a given collaborative association for use based on a system-dependent user interface. This would also involve creation of the object's meta-space and its corresponding meta objects.
	Delete	Removal of a collaborative object from the session (via a system-dependent user interface) so it is no longer available for use within any association.
	Manipulate	The collaborative object is "used" and/or performs its function. The user interface for this interaction is object-specific. This could also involve manipulation of the object's meta-space and its corresponding meta objects to modify differing meta-level aspects of the object (as outlined earlier). The interface for doing so would be system-dependent.
Behaviour	Add	Dynamic creation of a new behavioural pattern that can be used within a session to coordinate the activity within a collaborative grouping. How this would be done in terms of a user interface is system-dependent. This would also involve creation of the object's meta-space and its corresponding meta objects.
	Delete	Removal of a behavioural pattern so that is no longer available for use to guide entity activity. How this would be done in terms of a user interface is system-dependent.
	Manipulate	The manipulation of a behaviour would involve its real-time modification as the behaviour is being exercised (used) within the system. This could involve a system-provided user interface or be controlled automatically by the system internally. This would therefore involve manipulation of the behaviour's meta-space and its corresponding meta objects via reflection.

Table 3.7: Typical Application Phase Actions

type of implementation used to realize RASCAL class framework. This issue is further explained by example in upcoming chapters.

The conceptual and class frameworks therefore provides a customizable and tailorable skeleton on which to build specific RASCAL media space systems. How they are deployed in a distributed environment is discussed in the upcoming section on the RASCAL communications architecture.

3.2.5.3 The RASCAL Communications Architecture: Design, Function and Topology

The design of the RASCAL communication architecture is reasonably straight forward. Shown in Figure 3.34, the architecture's topology can be represented as a collection of hosts joined by a *coordinated communications subsystem* (abbreviated as *CCS*). While the traditional communications system is simply viewed as "passive" link between participants, the principle behind RASCAL is to provide an "active" communications infrastructure which "intelligently" manages media space activities. This "active intelligence" is provided by a specific part of the CCS known as the *collaborative multimedia support system*, or *CMSS*. The hosts form "points of presence" in the media space at which collaborative participants connect to the shared environment. Collaborative object (i.e. users, tools, media, etc.) are distributed at these points of presence and send/receive data and control information to/from other via the CCS. The CCS is intended to utilize network technology that supports multicasting (to facilitate simultaneous data delivery) and support for QoS guarantees, which are an important part of multimedia networking.

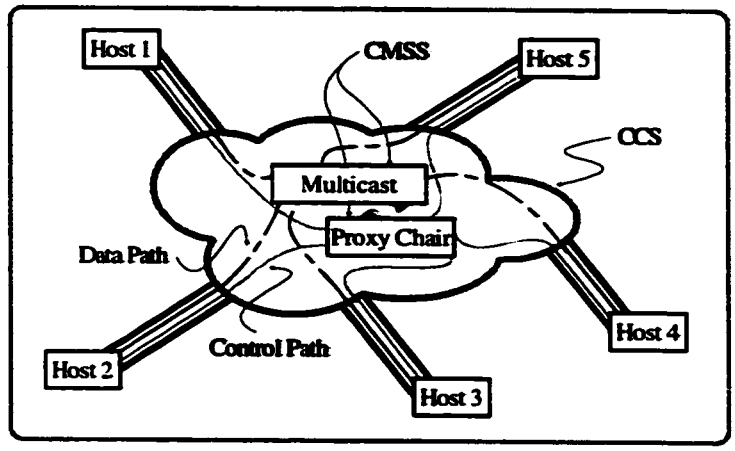


Figure 3.34: General Topology for a RASCAL System

A basic tenet of RASCAL's approach to behavioural management is the separation of coordination from execution. Shown previously in Figure 3.8, a RASCAL system is logically and complementary divided into two complementary halves: a *coordination sphere* and an *execution sphere*. Based on this architectural premise, the CCS is similarly divided into two distinct but complementary subsystems: the data subsystem and the control subsystem.

The execution sphere maps to the *data subsystem* which represents and deals with the flow of data that is produced and consumed by collaborative objects within the environment. The content and representation of the media, the tools and the users of the environment form the data that are multicast over the data portion of the CCS. Therefore, execution can be seen to constitute the "media" within a "media space". As stated earlier, such data is multicast to other hosts (within the bounds of a collaborative session/association) and can utilize support for QoS as provided by the underlying network (as specified by the appropriate entities QoS-related meta-objects). Within Figure 3.34, this connectivity is shown via the *data path*.

Coordination of media space data, on the other hand, is considered separately. Mapped to the coordination sphere, the *control subsystem* addresses the notion of behavioural management within the shared space. The coordination of execution (i.e. the coordination and synchronization of media, tool and user data) utilizes the *control path* within the CCS (Figure 3.34). The control information that describes the consumption of the data and events/actions performed in the space are transmitted along the control path, which like the media data, utilize support for multicasting and QoS. The key difference is that the flow of control information is regulated under the auspices of the CMSS.

To perform the regulation of control information, the CMSS logically pairs a *dedicated coordinator* with the multicast switch (i.e. the actual communications fabric). This top-level behavioural management entity is known as the *Proxy Chair* and manages control messages that describe media space data flowing through the multicast network. Such data itself is *not* channeled through the Proxy Chair, nor does the Proxy Chair perform any function other than coordinate the space through the regulation of control messages between source and

receiving hosts. From the collaborator's perspective, the network is providing the necessary coordination and synchronization, while from the network's point of view, the coordinator is a specialized entity within it that offers behavioural management services, akin to those associated with *intelligent networks* [ITU92].

Therefore, the CCS and the CMSS can be seen to collectively function as a "collaborative fabric", providing support for the various services required by collaborative environments based on the RASCAL framework. Through the use of reflection, the components of this "fabric" can "tailor" themselves to meet the ongoing needs of the media space as it is used over time. Figure 3.35 overlays RASCAL's conceptual framework entities on top of the communication architecture to illustrate their distribution in terms of a multi-session instance of a media space. Collaborative objects are shown at the host end-points while the collaborative behavioural definitions used for behavioural management (i.e. the collaborative associations) are shown as part of the CCS (in relation to the Proxy Chair). The behaviours of these collaborative objects are regulated via the use of collaborative behavioural patterns which also express a given entity's interactions with others. Therefore, it is these patterns that the Proxy Chair coordinates in order to regulate object activities and interactions within the space.

The use of specific communication protocols, QoS parameters and other traditional network considerations are considered specific to the entity in question and are dealt with as part of an entity's individual meta space. The framework and communication architecture imposes no specific requirements other than those already outlined; specific entities, however, may have their own specific requirements which an implementation will have to provide for. Such issues, including entity distribution and their usage will be discussed further in terms of an example media space scenario later in Chapter Four.

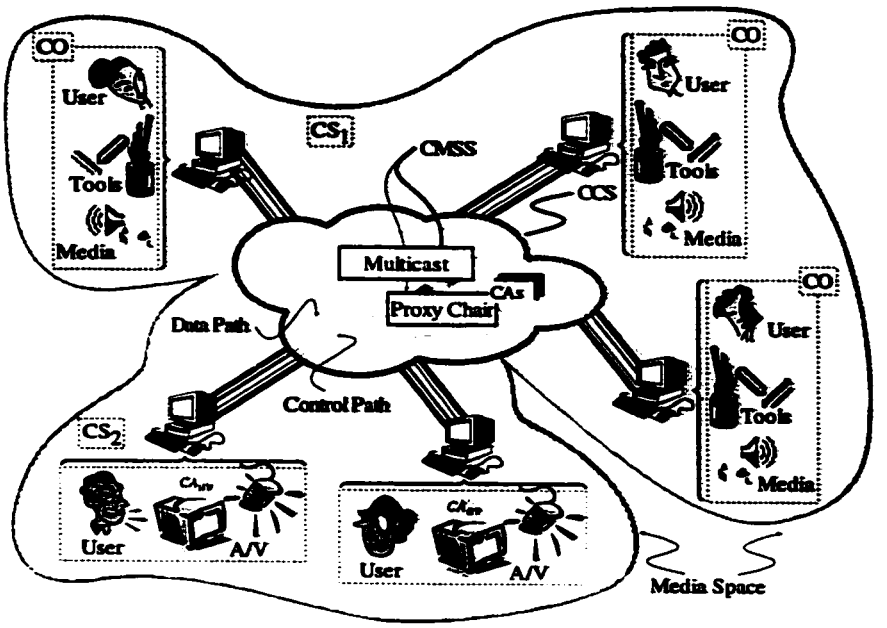


Figure 3.35: Relating Framework Entities to the Communication Architecture

3.2.5.3.1 The Global Perspective: The Proxy Chair

The Proxy Chair, often abbreviated as *Proxy*, is the top-most coordinator within any system based on the RASCAL framework. The Proxy Chair is so-named for its role as a "chairperson" which is "in charge" of globally regulating media space behavioural services, while still being independent of any application-specific management or "chairperson" entity (hence the "proxy" designation). In effect, it is a "global substitute" for behavioural management instead of requiring each application/host to address this issue internally.

The basic organization of the Proxy Chair is shown in Figure 3.36. As a coordinator, the Proxy monitors and controls all shared objects that constitute the media space itself. It consists primarily of three components:

Meta-Model: The meta-model is the specification of the current collaborative behavioural model being used to coordinate the space's activities. It is *not* a model of the current entities but a meta-level model by which the "reality" of the media space itself is to be compared.

Predictor & Analyzer: This component encompasses the analysis portion of the Proxy; in other words, it determines how the Proxy "compares" the actual behaviour of entities relative to the intended model and exercises decision over any adjustments that must be administered to entities within the space. Through appropriate use of the Proxy's database, it can also include logic that takes predictive action (such as resource allocation or synchronization adjustments) based on previous behaviours. The level of intelligence offered by such a service is widely variable and is related to the information held in the database (as discussed below), the expressiveness of the behavioural models and the support for precision and flexibility in their design.

Database: By utilizing a *Support Knowledge Base*, the Proxy can keep track of data important to meeting user needs and expectations. For example, the collaborative behavioural models that are used to coordinate and synchronize collaborative interactions can be stored in the database in a manner analogous to document templates for word processing packages. However, in order for the system to evolve over time, it must be aware of its prior operative history. The database therefore allows the

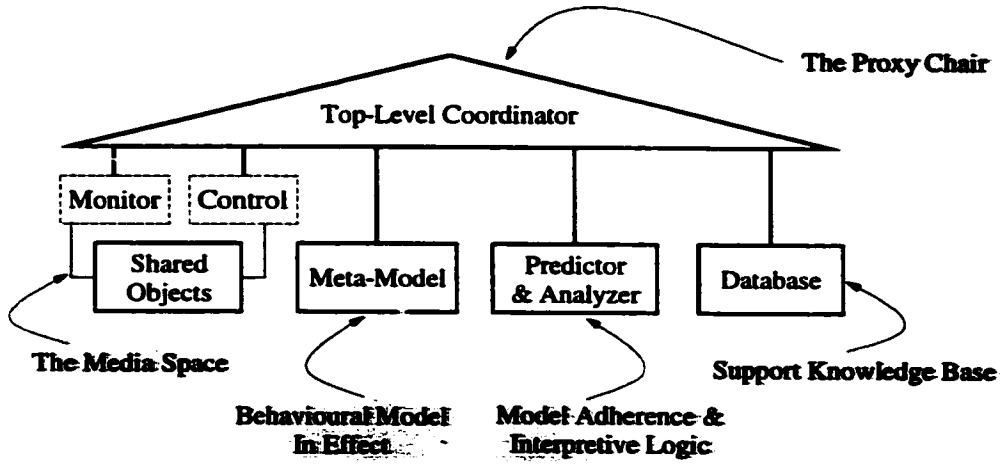


Figure 3.36: The Proxy Chair's Perspective

tracking of user behaviours, common behavioural patterns and significant system conditions that can be considered as part of performing resource management (based on media characteristics for example) and/or predictive processing (such as the adjustment of synchronization mechanisms to meet probable upcoming circumstances). Such information can persist over time (i.e. used across sessions) or transient (relative only to a single session). This notion of *Behavioural Histories* can be applied at varying granularities, such as scenario, group, user or object-specific. Implementation of the database itself is independent of the framework and can range from distributed/on-disk to centralized/memory-resident [OO97].

Because of its importance, the Proxy Chair is *not* a functionality to be assumed by a point-of-presence host; rather, it is to be a dedicated component of the CCS. Rather than waste resources, this separation offers the following benefits:

- A complete and straight-forward separation of concerns in terms of communication. The separation of paths enables appropriate QoS levels for different object data as well as control messages.
- A complete and straight-forward separation of concerns in terms of processing. End hosts will often be taxed by the manipulation of media data (such as decoding and user interface duties); therefore, placing the vital role of session behavioural management as a subservient task to such demanding (and often variable) tasks is ill-considered. More than the use of a centralized architecture, it is this often-assumed multiplexing of functionality which leads to bottlenecks; the temptation to minimize the number of components to perceive a more effective use of resources needs careful consideration.
- The autonomy of behavioural management facilitates is more straight-forward, scalable and flexible for both end systems and the Proxy. End hosts typically function as display and/or input devices for participants within the shared space. For specific applications of the shared space, the calibre of these machines could vary considerably. However, if considerable coordination logic is made part of the end host design, these machines necessarily become more complex and expensive. Furthermore, the potential orthogonality of host design is lessened because components are now context dependent as to their role within the media space; therefore, the adaptability and scalability of the architecture are now determined by application topology. As given, the architecture partitions system workload into two areas, each of which can be managed separately. The modification or addition of new coordination policies is more easily managed by using a central coordinator combined with host-independent, role-specific control software. While the Proxy forms a single, independent entity within the CCS, it is best viewed as a *logical* entity. Should media space scale, performance or failure be an issue, a group of coordinators could be defined to function as a *Distributed Proxy Chair*. In the case of failure, the Proxy could be dynamically replaced; or if the space scaled beyond the capacity of a single Proxy, multiple coordinators, each of which would manage a specific region of the media space, could be used to transparently migrate select sessions to another coordinator. That is, the Proxy could in fact be designed using cluster technologies if requirements warrant.

3.3 Summary

This chapter has presented the RASCAL framework as a means to facilitate construction of self-adaptive and evolutionary collaborative media space systems. The motivation and the rationale behind RASCAL, along with its basic conceptual structure and how it forms the basis for collaboration were presented. Additionally, the essence of the RASCAL methodology was given, including details on its concrete class structure and its support for the basic concepts introduced earlier. Finally, the underlying communications architecture was introduced and discussed in relation to the rest of the RASCAL framework.

Subsequent chapters will build on this information, illustrating its application to specific usage scenarios while addressing how certain aspects require special consideration in terms of real-world implementation.

4 CHAPTER FOUR

REALIZING RASCAL I: SCENARIO, APPLICATION AND UTILITY

This chapter augments the discussion of the RASCAL framework presented in Chapter Three by illustrating its application to various collaborative multimedia scenarios. As part of emphasizing the framework's flexibility, the scenarios range from the relatively practical to those that are more esoteric. An underlying theme of the framework and its illustrating scenarios is that they must account not only for present day needs but be able to facilitate those that while only envisioned at present, may well in fact be realized when necessary support technology becomes common place.

4.1 RASCAL Scenario Overview

The notion of a media space is inherently broad and open-ended. Consequently, its enabling and supporting infrastructure must be appropriately flexible and malleable so as not to restrict design or implementation; it must allow designers and users alike to realize both the requirements and the potential of a given collaborative scenario.

As an account of a possible course of actions or events [Web89i], a scenario illustrates the potential of a given kind of situation and is not an exhaustive description of all the possible activities that can occur. An electronic commerce system, for example, could provide an avatar-based virtual world for shopping at various retail outlets. This is only one possible interface for such a system and activities within the scenario could include interaction with sales staff, examining products as well as purchasing them. Similarly, in a tele-learning scenario, a distributed classroom environment could be utilized and encompass the large range of activities that occur in a teacher/student environment, from lecturing and discussion groups to taking examinations.

Therefore, while quite varied, the kind of situations suited to RASCAL can be characterized as follows:

- real-time, synchronous distributed environment
- activity/event orientation
- dynamic membership, group and activity definition
- diverse computational and collaborative multimedia requirements
- mixed behavioural coupling (i.e. tight to loose coordination/synchronization)

4.2 Sample Scenarios and Application Domains

This section details a number of collaborative scenarios and application domains that the RASCAL framework could enable and facilitate through the development of an appropriate media space. While not exhaustive, it is

intended to illustrate the range of possible uses as well as how the actual framework would map to specific situations. Different levels of detail are provided for the different scenarios to provide broad coverage.

4.2.1 Virtual Business Meeting

One popular use of telecollaboration technology is to facilitate business meetings across disparate geographical settings. Motivated by the potential of cost savings and flexibility in planning, these virtual meetings are typified by the traditional teleconference model in which users talk to and potentially see each other via audio/video connectivity. However, as presented earlier, the true collaborative potential of the meeting can require use and support for diversity in informational formats and ways to manipulate them. In other words, the ways people "meet" in a co-located setting must be facilitated within the virtual scenario. An important part of this is addressing how their desired activities and corresponding behaviours are enabled, supported and legitimized by the system.

For example, consider a meeting scenario of 10 participants which consists of the series of activities outlined in Table 4.1. This meeting effectively occurs in three coarse-grain segments: $lecture_1$, $discussion_1$, $lecture_2$. These segments of the meeting, however, can obviously be broken down into sets of specific actions. The $lecture_1$ segment maps to Table 4.1's step 2, which is in fact an ordered series of individual presentations (based on the notion of a pre-defined agenda). Similarly, the $lecture_2$ segment maps to step 3.4 but its exact sub-steps are based on the results from the $discussion_1$ segment (i.e. a dynamically defined agenda). For example, step 3.4 can be further decomposed as follows:

3.4.1 Regroup participants/brainstorming groups

3.4.2 Spokespersons for various brainstorming groups determined

3.4.3 Each spokesperson (in turn) presents their group's findings (using lecture style delivery)

Conversely, step 3.2 is deliberately left "open" since its constituent actions are depends on the meeting's context as well as the group, topic and participants in question.

Such a meeting is shown graphically in Figure 4.1 in which the various discussion (brainstorming) groups are mapped to different "locales" separate from the "main meeting area", which is shown in two states representing

Step	Description
1	Meeting participants convene/called to order
2	Presentational information by different members in well-known order (i.e. agenda)
3	Brainstorming period
3.1	Three groups of three "go to their corner"
3.2	Groups "discuss" specific issues as assigned by a facilitator
3.3	Throughout the above discussion, the group facilitator moves from group to group to assist and encourage interaction and development within each group
3.4	Groups reconvene and present discussion group results to the rest of the group
4	Meeting adjourns

Table 4.1: Example Meeting Steps

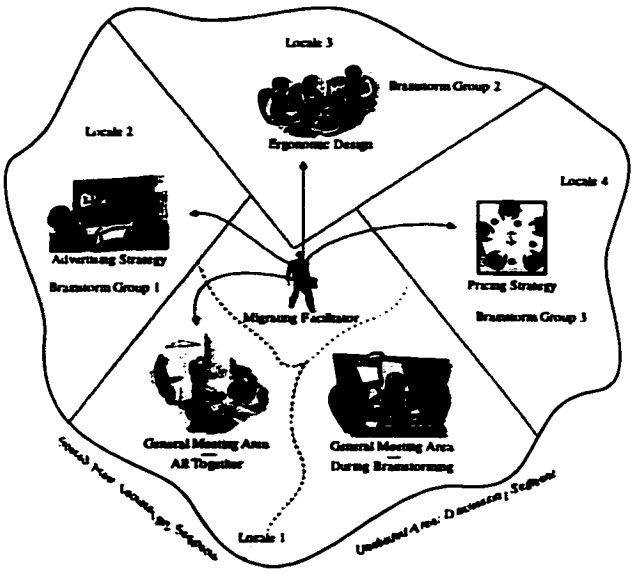


Figure 4.1: Conceptual Virtual Meeting Scenario

when everyone is gathered together and when the participants are separated into their groups. The groups include an advertising strategy group, an ergonomic/form factor design group and a pricing strategy group. The facilitator is shown as potentially migrating between the different locales as he "moves around the room" and interacts with each group in turn (Figure 4.2).

Moving from a co-located meeting to a virtual one utilizing a distributed environment necessitates mapping activities from one context to another. Table 4.2 illustrates one such possible mapping between them. Furthermore, the above activity breakdown and organization are purely one way of doing so; it is not the only, correct or necessarily best way to do so. Such issues are left to the meeting participants to exercise judgment on how the meeting is best organized and facilitated. Like the design of a meeting room wouldn't itself organize

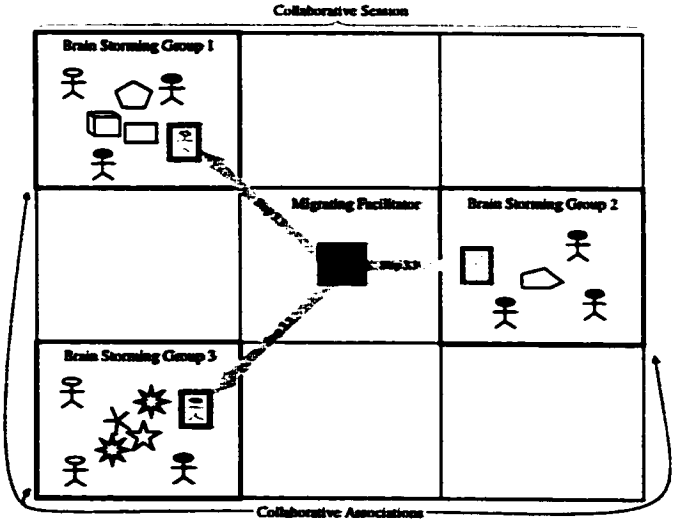


Figure 4.2: Example Virtual Meeting CSIG with Migrating Member

Step	Co-Located Scenario	Virtual Scenario
1	Call to order	Users join session
2	Series of presentations given by the meeting's participants	Users lead session using lecture style delivery in series
3	Set up brainstorming groups	Set up brainstorming groups
3.1	Physically isolate groups for privacy	User interface manipulated to segregate users into logical groups
3.2	Discuss and brainstorm	Discuss and brainstorm
3.3	Facilitator walks between groups and interacts with each of them separately	Facilitator manipulates user interface to interact with the formulated groups
3.4	Everyone gets back together and presents groups' results to everyone	Logical groups consolidate (via user interface manipulations) and resume lecture style delivery
4	Adjournment	Users quit/leave session

Table 4.2: Stepwise Comparison of Co-Located and Virtual Meeting Scenario

the actual activities within a co-located meeting, neither should an enabling framework limit organizational freedom within a virtual one.

In terms of the RASCAL framework, each brainstorming group is an instance of a Collaborative Association, since it is a grouping of entities that are actively "working together" in some semantically defined relationship. As the groupings of who is working together changes, the collaborative associations will change. Those working together in a specific CA will be coordinated according to a behavioural specification that deals with the intended purpose and characteristics of that group. Each CA can therefore be thought of having its own sub-goal which is related in some manner to achieving the top-level goal associated with the collaborative session. Table 4.3 shows the various goals and the CAs associated with them, including those which directly map to the different discussion groups (advertising strategy CA, ergonomic/form factor design CA, pricing strategy CA). The "main meeting area" is part of the overall meeting goal and therefore associated with the top-level CA that is part of the scenario definition. The facilitator migrates between CAs dynamically as he "moves around the room" and interacts with each group in turn. The pricing strategy CA is detailed in Figure 4.3 to illustrate specific framework constructs. An example of utilizing the CARE meta space is shown in Figure 4.4 such that the synchronization paradigm of the pricing document could be variable and is part of the editing semantic of the CA. For example, editing could be restricted to a single user at a time using a conservative locking scheme or an optimistic versioning/resolution approach could be taken so that each user edits their own version of the document and differences are resolved in a context-dependent manner (provided as part of the CARE meta space implementation).

Goal Level	Collaborative Association
Overall Meeting Goal	Finalize product launch plans
Brainstorming Groups' Subgoals	Advertising strategy
	Ergonomic/form factor design
	Pricing stategy

Table 4.3: Goal to Collaborative Association Mapping

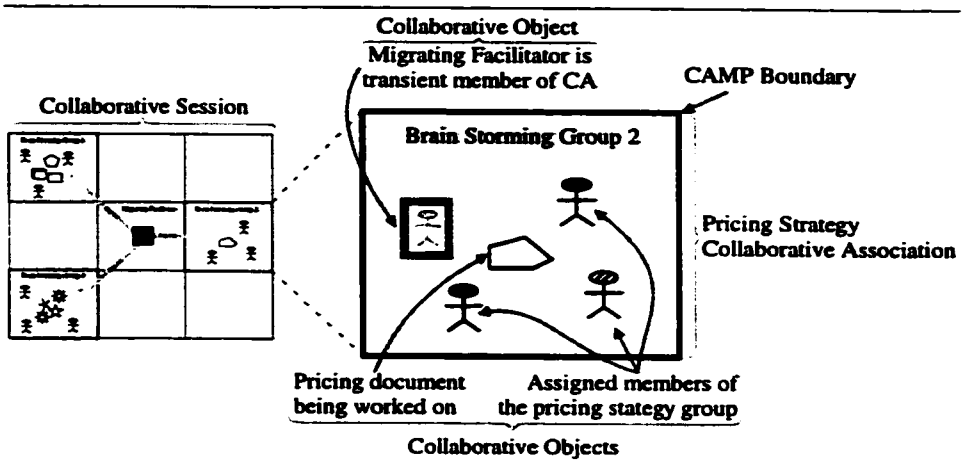


Figure 4.3: Example Meeting CA in Detail

4.2.2 Tele-Learning

An oft-cited benefit of combining modern communications and multimedia technology is the potential afforded distance education. Supporting the educational process can be a challenge as the typical classroom is quite often a myriad of different types of behaviours which change over time. Depending on the content of what is being taught, the instructor and the students, different instructional techniques and interactions are used at different times and under different circumstances.. Therefore, such a scenario provides a good example of how a RASCAL media space can facilitate a flexible collaborative environment.

4.2.2.1 General Tele-Learning Topology

A typical tele-learning system will consist of multiple students and a single instructor interconnected by a communications network. Sometimes, special educational software is used along with general purpose applications and tools to facilitate different kinds of interaction. Audio/video conferencing is quite common but shared applications and use of "canned presentations" are also increasingly part of the educational process.

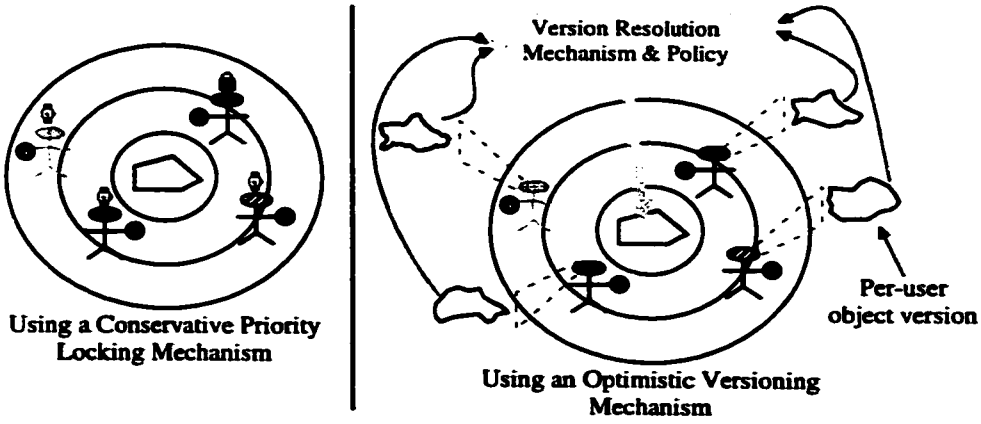


Figure 4.4: Possible Synchronization Paradigms for the Pricing Strategy CA

Such an environment therefore can be seen to constitute a collaborative media space through the use of real-time audio/video capture/display in combination with support for applications, like a shared whiteboard, the distributed playback of synthetic media such as video clips, animated presentations, and so forth.

Unfortunately, despite having the logical character of a media space, many traditional tele-learning systems function either as an uncoordinated collection of software packages (such as those based on the MBONE suite) or those that place role-specific computational demands on select elements of the topology. For example, the instructor's machine is often the most powerful and is designed/constructed with specific requirements by virtue of its role. Machines which represent students within such an architecture are typically less capable and limited in functionality because of their intended role within the tele-learning system. The difficulty with such an approach is the rigidity in how it is used, which is imposed by virtue of its initial design, rather than how the users wishes to configure it. For example, in the traditional approach, if control for coordination of the classroom is embedded in software on the instructor's machine, it intrinsically disallows students the ability to lead the classroom, as may be desired by the instructor at some point in time. The result is a topology that is very static and uneven in terms of its hardware components, software distribution, managerial, functional and ultimately collaborative potential.

In the RASCAL approach to a tele-learning media space, the primary coordination logic is contained in the CCS (i.e. network), rather than in specific end-hosts (as shown in Figure 4.5). As the traditional method has coordination and synchronization logic placed in highly role-specific entities, the standard method results in a more complex and less orthogonal system where functionality is determined more by hardware and topological assumptions rather than software control mechanisms. The RASCAL model promotes an orthogonal system in which coordination is provided by a coordinated communications subsystem under the control of host-independent software. The instructor's workstation, rather than be a single, complex and expensive machine, can be a "simple" entity with an interface to the behavioural management logic within the CCS. By increasing

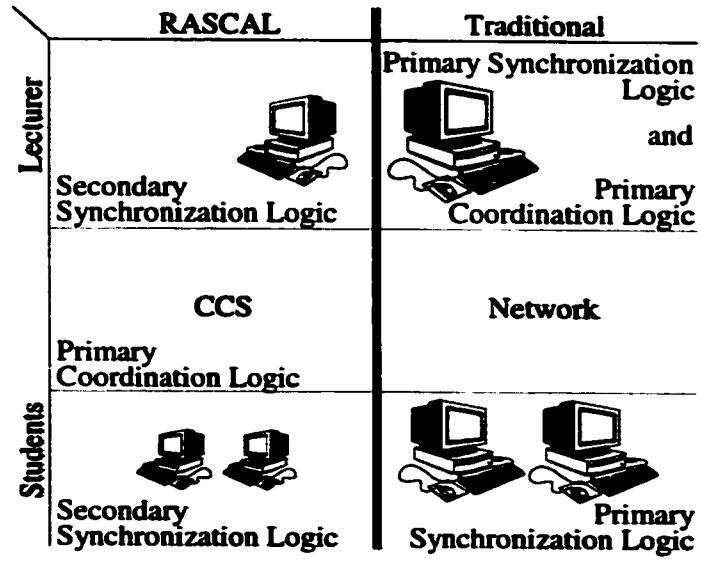


Figure 4.5: Comparing TeleLearning Architectures

host equity (reducing differences in calibre between student and teacher machine requirements), the system's instructional policy can be enforced by role-based control software, rather than hardware layout. The ability to reduce the impact of resource requirements on student machines is also important given the burgeoning trend to low-cost network appliances such as set-top boxes or digital entertainment terminals (DETs). Consequently, by taking a network services approach, topology and functionality are more orthogonal and changes to either are easier. Specifically, the architecture offers more flexibility in the way that students can be involved within the media space. Since a participant's role is not determined by the complexity of coordination logic/mechanisms in their local host, the CCS allows both students and instructor to participate in or lead the "classroom" based on policy, not topology.

Therefore, the priority and management hierarchy of the media space is determined by control software at individual hosts which provide a management interface to the CCS. This provides more flexibility in development and extension of the system by encouraging high-level configuration of the media space using adaptable, policy-based software rather than application-dependent hardware. While end-hosts may require some secondary synchronization logic (as discussed later), coordination is primarily achieved through the intelligent communications system which is a more convenient and topological-independent location for policy-enabling mechanisms.

The CMSS is the part of the CCS which actually provides the policy-based behavioural management on behalf of the network. The main part of the CMSS is the Proxy Chair, which functions as a coordinator and is responsible for the space's behavioural management according to its management policy. It is important to note that the coordinator (i.e. Proxy Chair) does *not* represent a domain-dependent manager (i.e. the instructor or the business meeting facilitator in the previous example) but an entity which enforces the guidelines of the space for all participants within it. Therefore, in terms of a virtual classroom, the coordinator guides the instructor as well as the students; in terms of the virtual meeting, the coordinator guides the facilitator in addition to the regular meeting participants. Each of these entities are collaborative objects within the media space which are regulated according to the behavioural patterns associated with their particular interactions with other entities. It is these patterns that the Proxy Chair coordinates in order to regulate activities within the space and therefore, the collaboration in question.

4.2.2.2 Unsupervised Single Classroom Scenario

As an initial example, consider the collaborative scenario outlined in Figure 4.6, which maps the media space to an unsupervised virtual classroom. In this scenario, two well-known activity groups and one passive audience group are shown. Each of these groups form a CA within the scenario as shown in Figure 4.7 and are detailed in Figure 4.8 to illustrate specific framework constructs. These groups include: (1) two participants jointly editing a document with live audio/visual conferencing; and (2) an overlay annotation being drawn over a movie presentation. In addition to these members of the telelearning scenario that are actively collaborating, other students can be passive observers in the classroom (i.e. inactive members of the session) and would not be

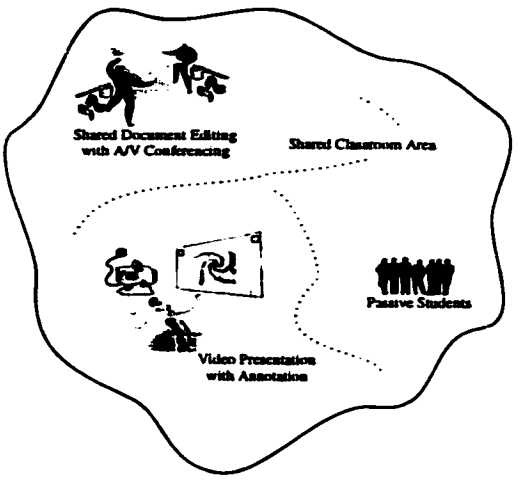


Figure 4.6: Unsupervised Single Room TeleLearning Scenario

part of a specific well-known collaborative association (i.e. they would be a member of the default Null CA) as they would not be "working together" in a semantically defined relationship yet would still probably want to know what was going on in the classroom. This is itemized in Figure 4.9 and shown in relation to the CA allocation in Figure 4.10 resulting in the collaborative behaviour tree shown in Figure 4.11. Entity correspondence across Figure 4.9, Figure 4.10, Figure 4.11 and Figure 4.12 is shown via the circular numerical designations.

Within this scenario, the highest meta level is a global "classroom policy" which guides all interactions in the classroom, such as regulations as to how individuals interact, what kinds of group activities are allowed (such as brain-storming or working on a shared whiteboard) and so forth. These policies and their related mechanisms are form patterns (shown in Figure 4.12) which specify the role and interaction between entities within the media space. The Proxy Chair serves as a global coordinator which manages the coordination of the

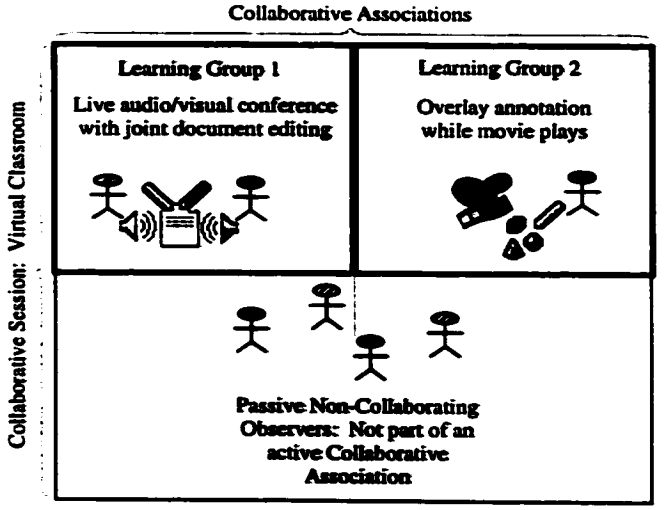


Figure 4.7: Unsupervised Single Classroom CSIG

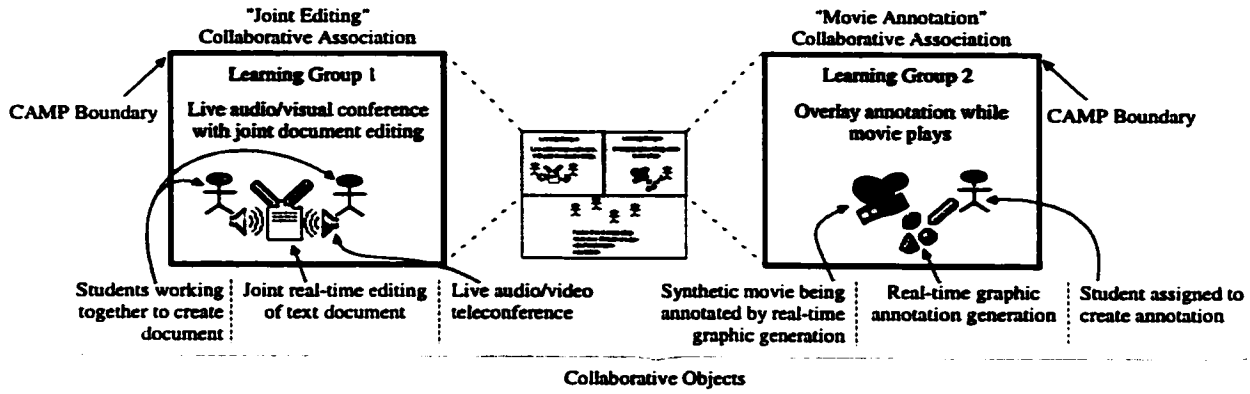


Figure 4.8: Detailed Example of Unsupervised Single Classroom CAs

entire space (the classroom) according to its management policy (the classroom policy). It is important to note that the Proxy does *not* represent the instructor but an entity which deals with the behaviour of all participants within the space; therefore, the Proxy would regulate the instructor's behaviour as well as that of the students.

While the classroom policy forms the meta level for the coordinator, the coordinator is itself meta to the collaborating groups. The coordinator governs the coarse-grain behaviour of the individual groups using the classroom policy while individual groups are governed in a similar manner but at a finer granularity and using their CA's behavioural pattern. For example, while the maintenance of audio/video "lip-sync" is done on a per pair basis, it is coordinated with respect to sibling collaborators under the control of the coordinator using a "conversation" behavioural pattern. This non-monolithic approach enables different paradigms and

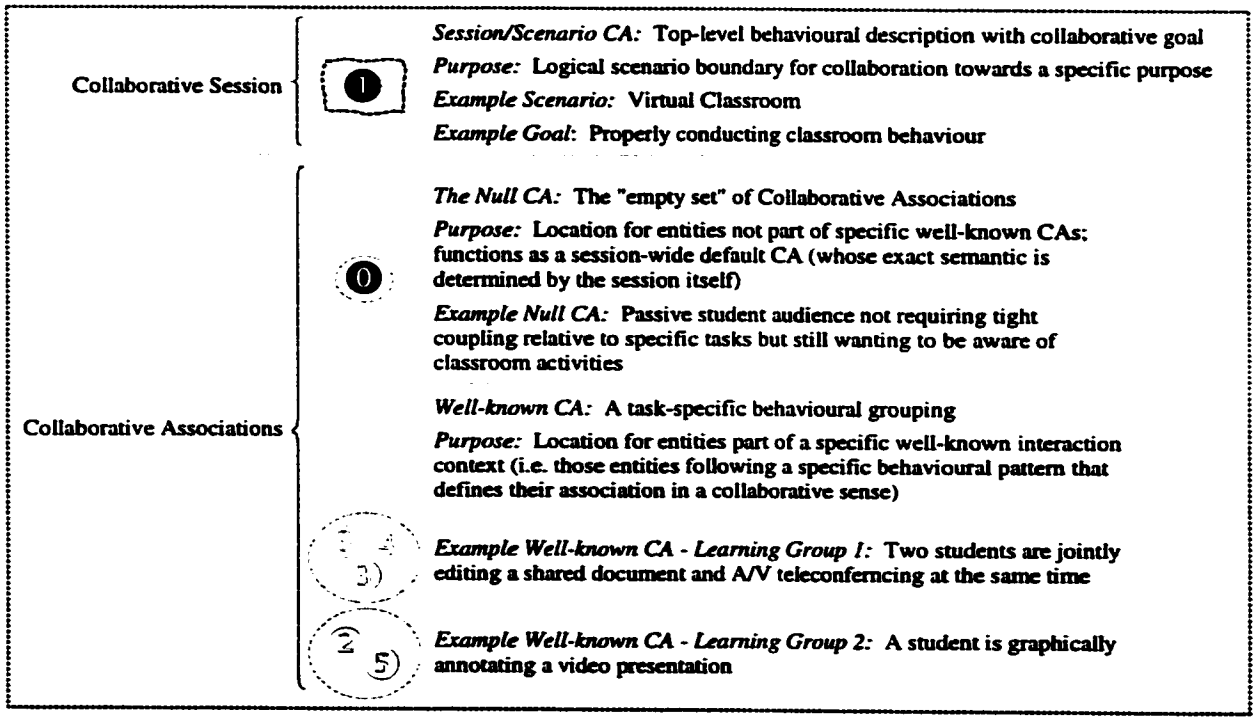


Figure 4.9: Single Classroom Session and Associations

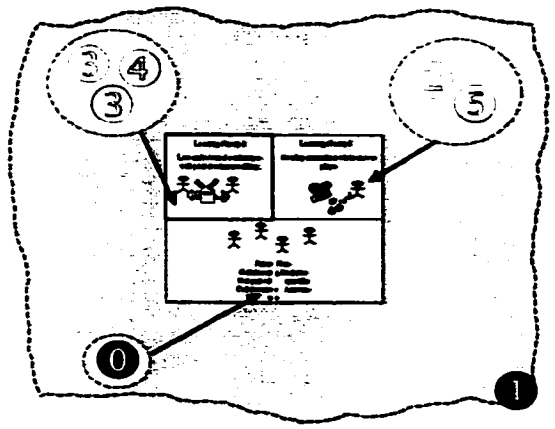


Figure 4.10: Unsupervised Classroom CAAD

granularities of synchronization to be specified as part of a specific group's local synchronization policy. Additionally, the appropriate considerations for mixed mode operations (how live and synthetic media are combined) can also be dealt with in a similar manner (see upcoming section on the application of RASCAL to multimedia synchronization). And finally, the mechanisms which actually perform the operations are at the lowest level in the hierarchy and represent the media-specific execution sphere. Consequently, complex classroom activities can be broken down into smaller subsets of policies and mechanisms, independently managed while still globally related to the overall classroom goal/semantic.

The tailorability offered by using such an approach means that different educational requirements can be met by integrating different components (i.e. specific demonstration applets or informational material, such as an on-line textbook). It also offers support for a wide range of classroom settings and learning styles, such as specifying interactions between students and the tools they can use. Such components could conceivably be intelligently modified or replaced under the control of the system itself (based on self-investigative meta-computations by higher-level entities). The ability to monitor and adapt via the application of reflection would effectively enable the classroom to dynamically "learn" how best to support its students/instructors based on their usage characteristics. Possibilities include changes in synchronization granularity according to user satisfaction with the provided quality of service, switching to a less stringent synchronization paradigm based on frequency of access and the dynamic reconfiguration of how collaborative objects are related (i.e. who is working together as part of the collaborative groupings).

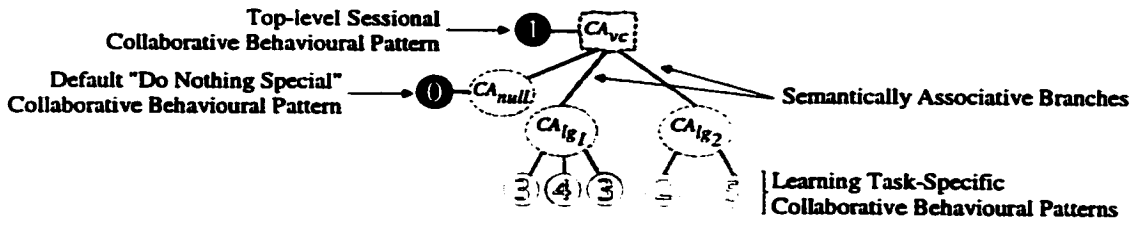


Figure 4.11: Unsupervised Single Classroom Scenario CBT

4.2.2.3 Supervised Multi-Classroom Scenario

As a second example, consider the more involved example of a tele-learning scenario outlined in Figure 4.13, which maps the media space to a set of virtual classrooms with the addition of an instructor that moves around between them. In this scenario, three well-known activity groups, one passive audience group and a migrating instructor are shown. Each of these groups form a CA within the scenario as shown in Figure 4.14 and are detailed in Figure 4.15 to illustrate specific framework constructs. These collaborative groups include those outlined in the previous unsupervised scenario but are augmented by a third well-known collaborative group which manages student examinations (which are logically in another classroom, away from the main classroom interaction). In addition to these members of the telelearning scenario that are actively collaborating, other students can be passive observers in the classroom (i.e. inactive members of the session) and would not be part of a specific well-known collaborative association (i.e. they would be a member of the default Null CA) as they would not be "working together" in a semantically defined relationship yet would still probably want to know what was going on in the classroom. The instructor is shown as a top-level collaborative object (initially created as a member of the scenario-level CA) which is potentially related (in a logical sense) to all the different activities within the entire space. This is itemized in Figure 4.17 and shown in relation to the CA allocation in Figure 4.16; the resulting collaborative behaviour tree shown in Figure 4.18. Entity correspondence across Figure 4.16, Figure 4.17, Figure 4.18 and Figure 4.19 is shown via the circular numerical designations.

As with the previous example, the highest meta level is a global "classroom policy" which guides all interactions in the classroom, such as regulations as to how individuals interact, what kinds of group activities

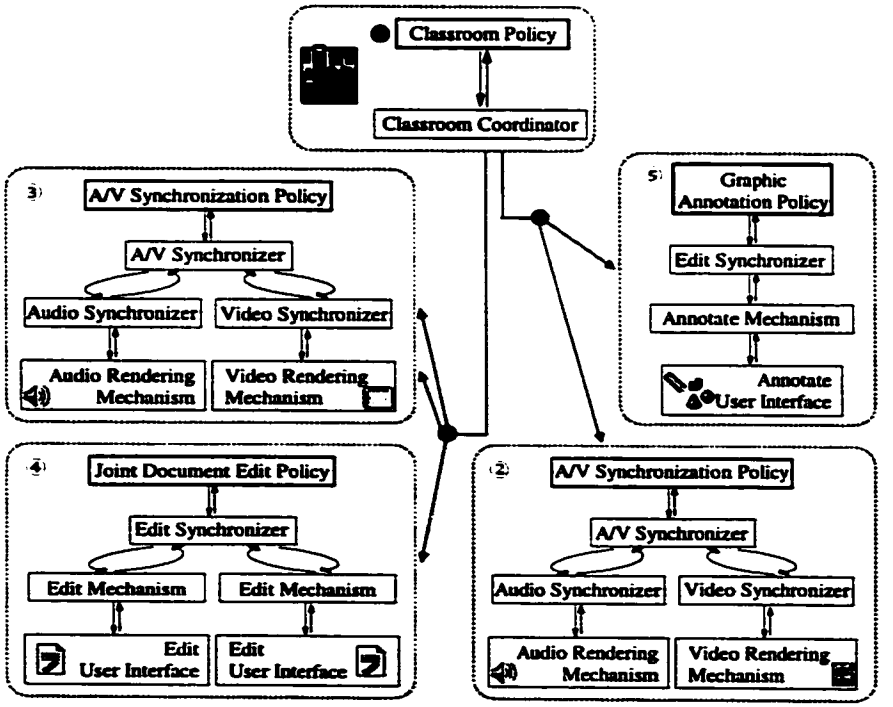


Figure 4.12: Unsupervised Single Classroom CAPRE

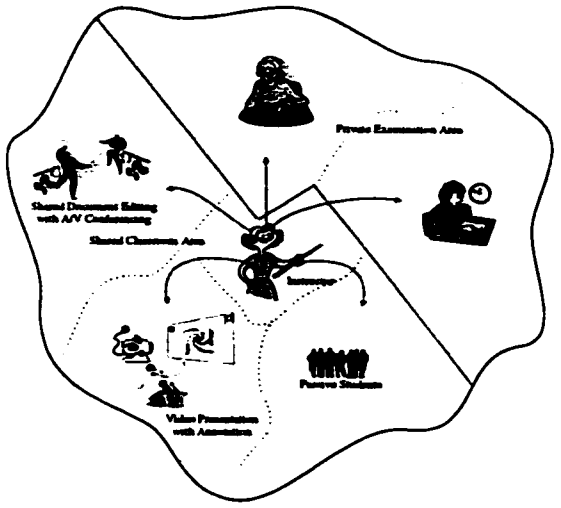


Figure 4.13: Multi-Room Telelearning Scenario

are allowed (such as brainstorming or working on a shared whiteboard) and so forth. These policies and their related mechanisms are form patterns (shown in Figure 4.12) which specify the role and interaction between entities within the media space. An additional top-level "instructor" object is initially managed via the "classroom policy" as it can potentially interact with any other entity in the classroom(s). As an independent entity that can migrate between the learning and exam groups within the environment, the instructor is analogous to "facilitator" within the product launch example given previously.

As the instructor migrates between different groups, it becomes a temporary member of the appropriate collaborative associations. When not performing any particular interaction with a group of students, the instructor collaborative object would typically revert back to its scenario level CA (a reasonable design for most tele-learning scenarios). As before, the Proxy Chair serves as a global coordinator which manages the coordination of the entire space (the classroom) according to its management policy (the classroom policy). It

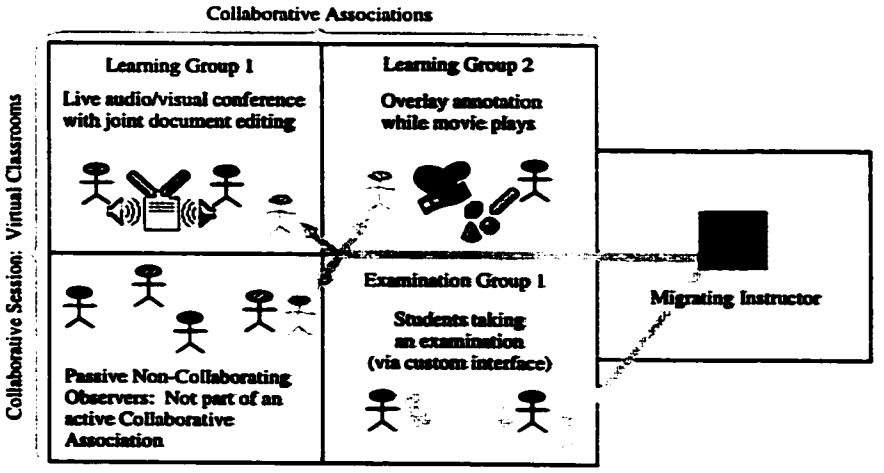


Figure 4.14: Supervised Multi-Classroom CSIG

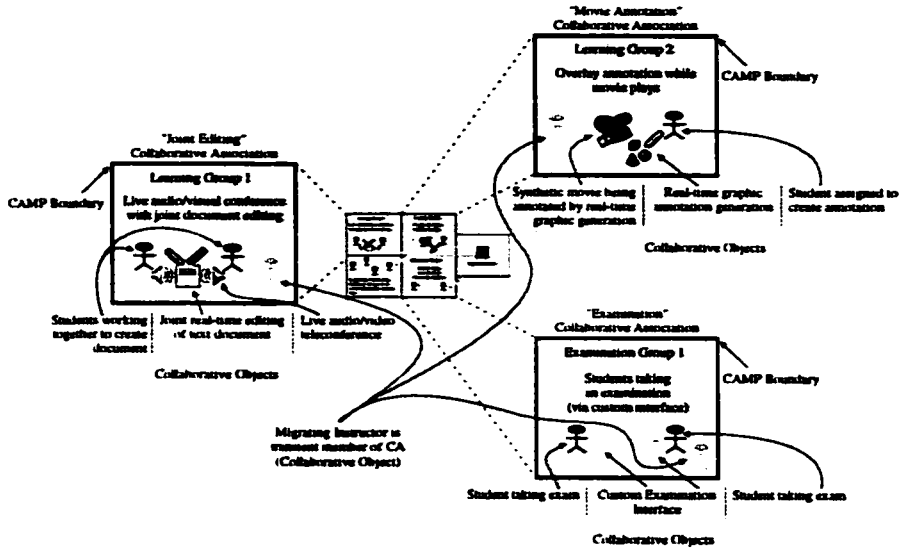


Figure 4.15: Detailed Example of Supervised Multi-Classroom CAs

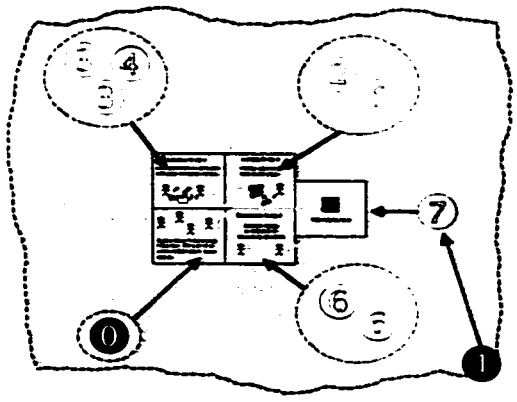


Figure 4.16: Supervised Multi-Classroom CAAD

is important to note that the Proxy does *not* represent the instructor but an entity which deals with the behaviour of all participants within the space; therefore, the Proxy would regulate the instructor's behaviour as well as that of the students.

The tailorability offered by using such an approach means that a scenario such as this one can orthogonally model different education environments (single vs. multiple classroom) and integrate different levels of management and dynamic collaborative association groupings as simple extensions to basic collaborative behavioural models. While the implementation of the interface and control logic of the permissions maybe somewhat involved for such a scenario, they form part of the collaborative association's meta space and can therefore be addressed independently of the basic organization. For example, an interesting possibility would include the notion of a "reflectively autonomous" instructor that adapts its instructional behaviour to meet the needs of its students. Rather than have a human user represented by an avatar in a collaborative virtual environment, the behaviour of the instructor could be controlled by an autonomous intelligent agent that

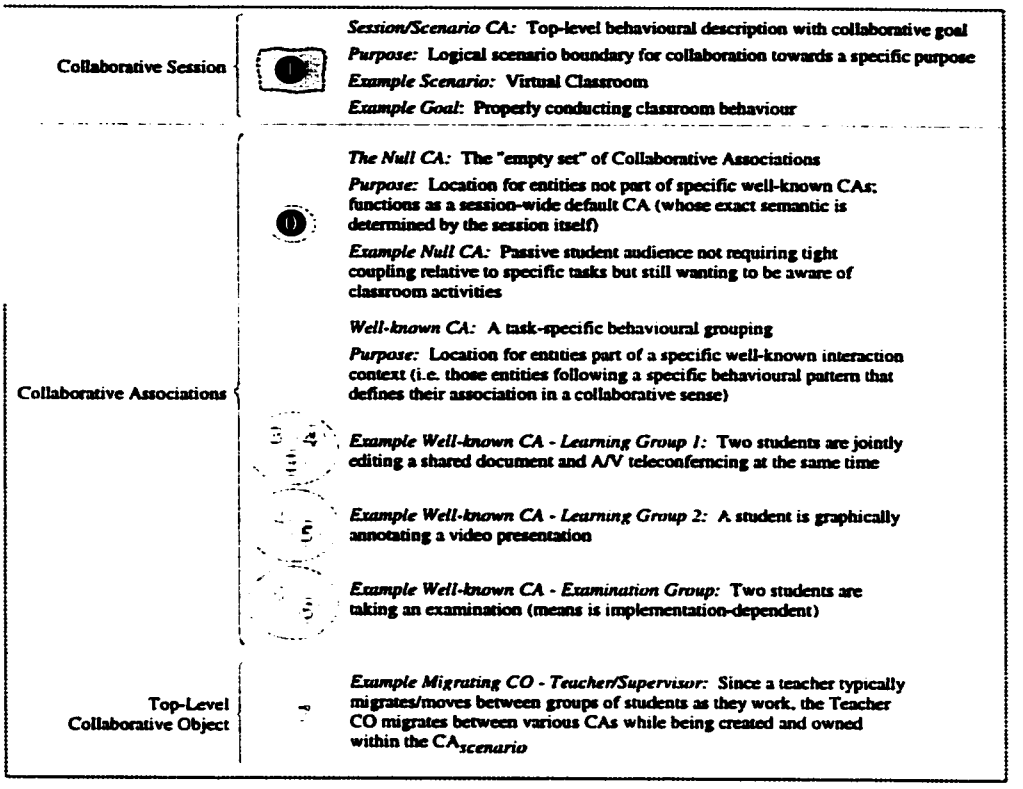


Figure 4.17: Supervised Multi-Classroom Session and Associations

dynamically adapts its supervisory rules for different groups of students. In some senses, this is analogous to a "substitute teacher" managing the classroom (i.e. not the real teacher) according to the real teacher's instructions (its basic supervisory rules) but it adapts what actually goes on in the classroom relative to actual reality of the students, their behaviour and the needs of the situation (i.e. it reflectively adapts itself).

4.2.3 The Virtual Arts: Music, Theatre and Dance

The artistic arena has always been a source of inspiration for what could be accomplished with multimedia technology. Systems that support artistic ventures, however, are very often special purpose and offer very little common ground with more traditional applications. For example, very seldom do projects address joint document editing and distributed music rehearsal in the same work. However, because of RASCAL's emphasis

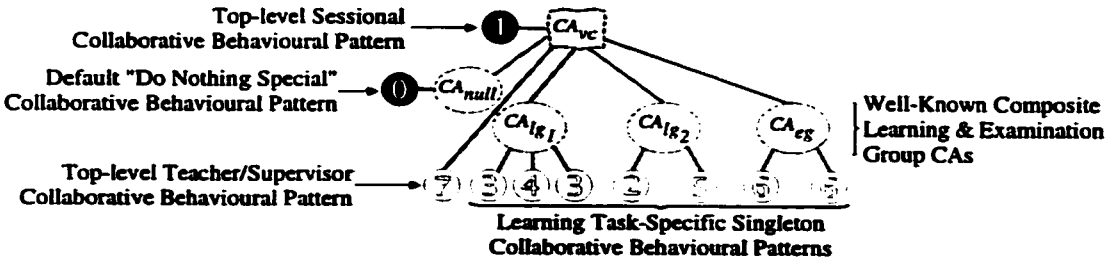


Figure 4.18: Multi-Room TeleLearning Scenario's CBT

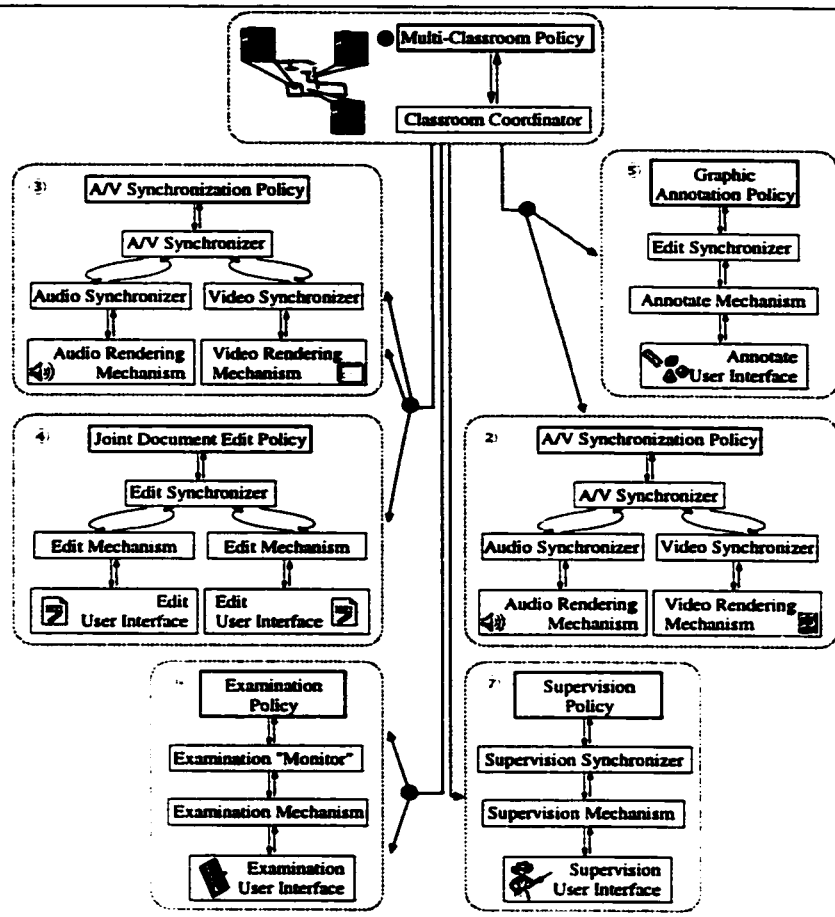


Figure 4.19: Supervised Multi-Classroom CAPRE

on dynamic behaviours , the same framework can orthogonally be applied to both practical and more esoteric pursuits. As an example, a RASCAL media space is now shown facilitating three different artistic scenarios.

4.2.3.1 Tele-Orchestra

By facilitating interaction between geographically separated musicians, a tele-orchestra scenario offers the opportunity to perform and rehearse in a non-traditional manner. Musicians that need to rehearse for an upcoming concert could do so while located in different cities, using a media space as a "virtual concert hall". Such an example is shown in Figure 4.20 in which three musicians are practicing under the tutelage of a conductor. In the simplest case, since each musician at a different locale forms its own collaborative association, as does the conductor. Within this example, there is a different combination of collaborative objects amongst the various CAs as the example does not provide complete orthogonality amongst all the participants; that is, not everyone has the same media/information flows available to them. In particular, the sound of each participant is multicast to the others, as each needs to hear the other in order to have a sense of awareness (in terms of musical performance). However, while each of the instrumentalists needs to see the conductor (hence having video reception), the conductor does not (strictly) need to see the instrumentalists and

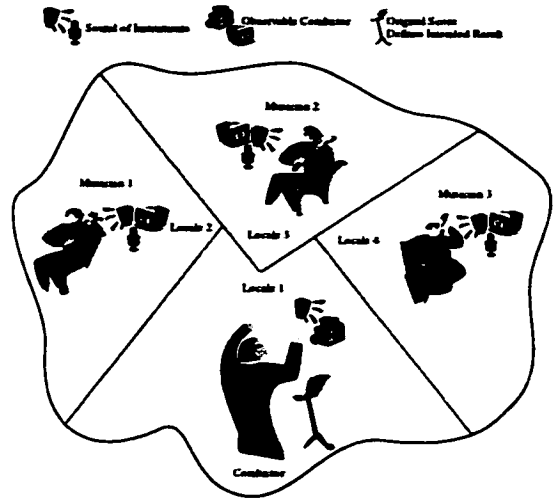


Figure 4.20: Basic Tele-Orchestra Scenario

therefore, has video capture while the players do not. In a different configuration, full-duplex video between the conductor and the instrumentalists could be used as could a configuration where full-duplex a/v between instrumentalists without the use of the conductor could provide a more peer-to-peer (*aka* jam session) configuration of a distributed orchestra. All three of these configurations are illustrated in Figure 4.21. Naturally, the size of the group and its particular needs (in terms of how the musicians "work") would influence the actual design and configuration of the media space. The flexibility of RASCAL, however, allows the same basic framework to be able to support such a wide-range of configurations (see Table 4.4).

The use of different CA configurations enables the same basic scenario to be exploited and managed in a

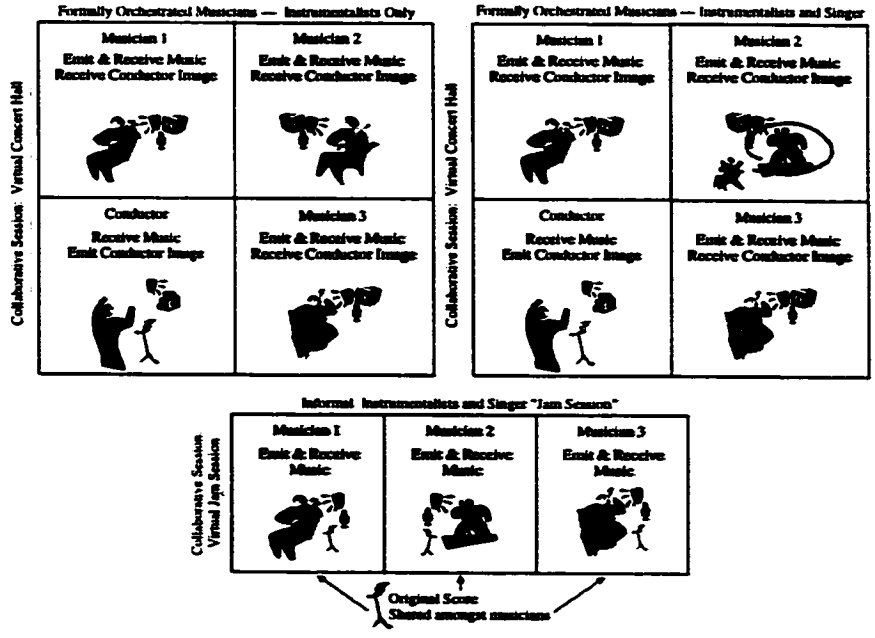


Figure 4.21: Possible Tele-Orchestra CSIG Variants

<table>
<tr><th colspan="2" rowspan="2">Framework Entity</th><th colspan="3">Scenario Variant</th></tr>
<tr><th colspan="2">Formal Instrumental
Configuration 1 | Configuration 2</th><th>Formal Singer</th><th>Informal Jam</th></tr>
<tr><td colspan="2">Collaborative Session</td><td colspan="2">Virtual Concert Hall</td><td>Virtual Jam Session</td></tr>
<tr><td rowspan="3">Collaborative Association</td><td>Composite</td><td></td><td>Instrumentalist</td><td colspan="2">Instrumentalist, Vocal</td></tr>
<tr><td>Singleton</td><td colspan="3">Conductor, Musician 1, Musician 2, Musician 3</td><td>Musician 1, Musician 2, Musician 3</td></tr>
<tr><td>Null</td><td colspan="4">Not used, since all entities have well-defined tasks; could be defined as audio receiver if the notion of an audience was allowed</td></tr>
<tr><td colspan="2">Collaborative Objects</td><td colspan="3">Audio capture & emission, Video capture & display</td><td>Audio capture & emission</td></tr>
</table>

Table 4.4: Framework Elements per Tele-Orchestra Scenario Variant

variety of ways. Detailed in Figure 4.22, Figure 4.23 and Figure 4.24, these "variations on theme" therefore benefit from the ability to re-use functional entities as well as behavioural patterns. For example, the formal instrumental virtual concert hall is shown with two configurations: (1) each collaborator is in their own independent CA that is managed by the overall collaborative session; and (2) the conductor CA is managed directly by the collaborative session but the three musicians are grouped together in an instrumentalist CA which manages them as a group with respect to the session. The benefit from the second configuration is that different behavioural characteristics (such as tighter audio synchronization and QoS) could be requested for the instrumentalists en masse, whereas that for the conductor can be adjusted independently. Furthermore, additional musicians do not directly impact the scenario definition in the second case but are dealt with at in terms of the instrumentalist CA. This is an example of how scalability can be addressed through the appropriate levels of behavioural indirection. Should a full orchestra of many different types of instruments be required, the instrumentalist CA could have child CAs that address specific types of instruments (which then could be addressed in terms of behavioural management and characteristics as a group); for example, woodwinds, brass, percussion, strings and so forth. While the basic behavioural pattern would be re-used amongst the different groups, it could be customized for each via its CARE meta space management interface (by the conductor). For example, when a specific section would not be playing their instruments for a period of time, resources required by them could be allocated to others.

The concert hall variant that utilizes a singer instead of a instrumentalist (detailed in Figure 4.25) is also an example of behavioural re-use. Since the "basics" of a music behavioural pattern would be shared amongst any type of musician, the same "musician-level" singleton CA could be used to represent the singer. However, it would be managed by a vocal composite CA which would then manage and control any differences in

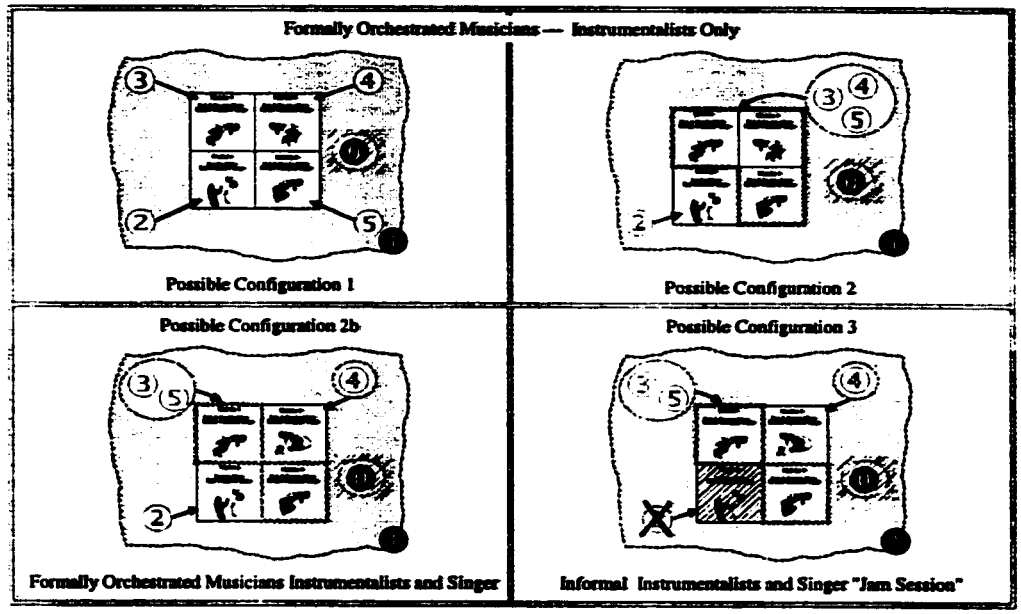


Figure 4.22: Tele-Orchestra CAAD Variants

performance characteristics and requirements between the instrumentalists and that of the singers. Now, the conductor, the instrumentalist and the vocal CAs are directly interacting via the collaborative session and group-dependent management is done by the appropriate composite CA. Because each of these could expose a management interface, they are then of course open to manipulation by another entity in the session, such as the conductor.

Within the musical context, there is the need to coordinate the participants' actions in terms of order and

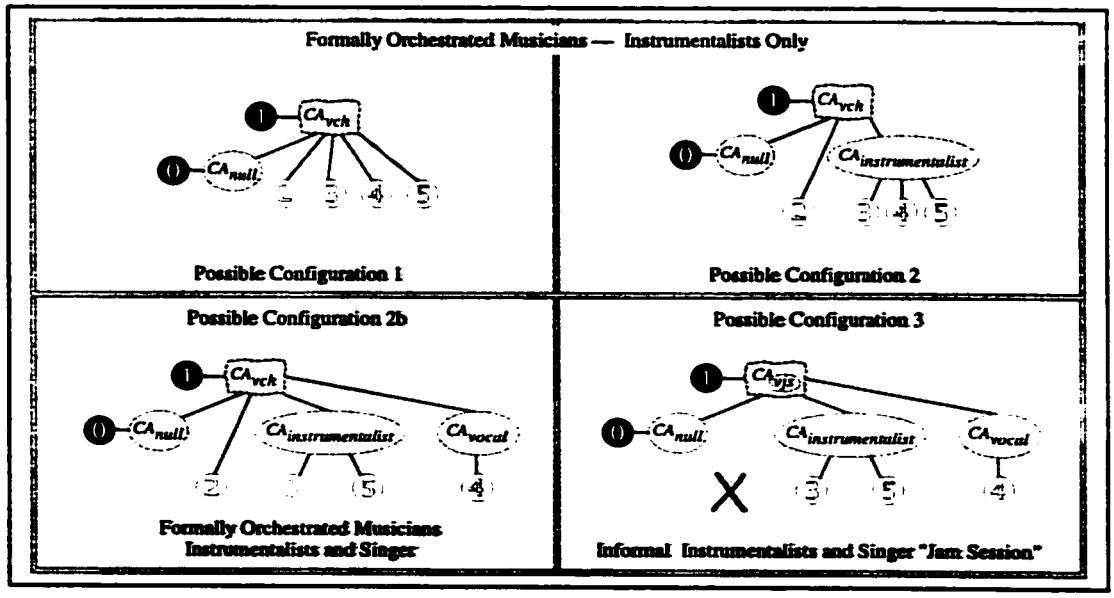


Figure 4.23: Tele-Orchestra CBT Variants

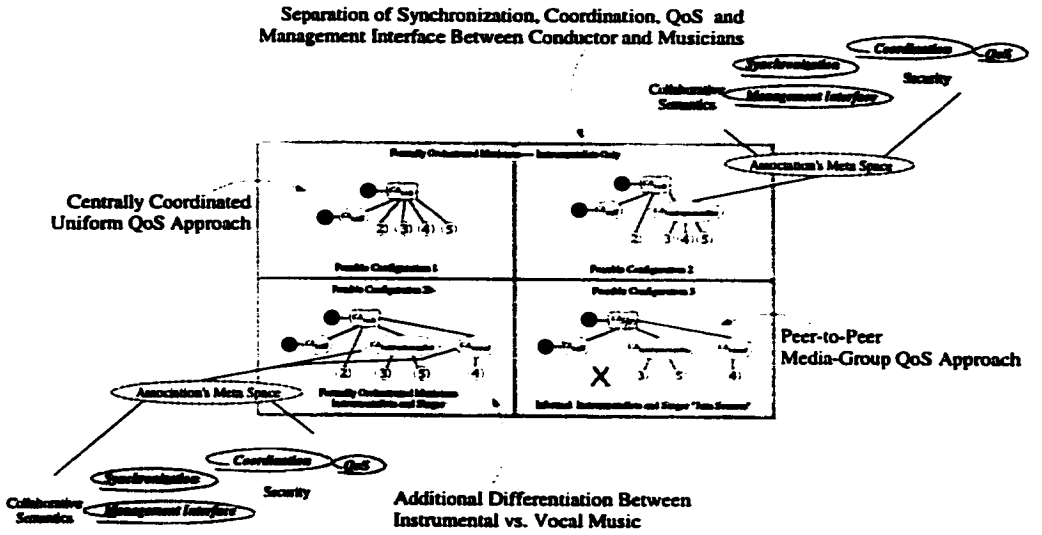


Figure 4.24: Annotated Tele-Orchestra CBT Variants with Meta Spaces

timeliness. Additionally, an awareness of the other musician's behaviours as well as that of any conductor must be maintained so each participant can adapt themselves to the reality of a given performance. This includes subtle variations indicated by the conductor as a result of musical interpretation (*aka* artistic license).

Finally, the informal jam session is much like the formal singer variation except for the omission of the conductor. Therefore, because of the design of the scenario, there is no longer a need for video capture and display (as it was used solely with respect to the conductor). Therefore, the actual objects used within the space now are simplified as well. And as before, because management interfaces can be exposed for manipulation by another entity in the session, such as the system itself, automated QoS adjustment/allocation based on system knowledge of the musical score and its "performance requirements" (i.e. singer doesn't sing for a segment of the whole song, in another section he/she is *a cappella* – hence synchrony requirements/QoS can be adjusted/relaxed during these segments if system needs resources elsewhere). Artistic interpretation of the score can be stored as behavioural history for the given artists in combination if they regularly perform the

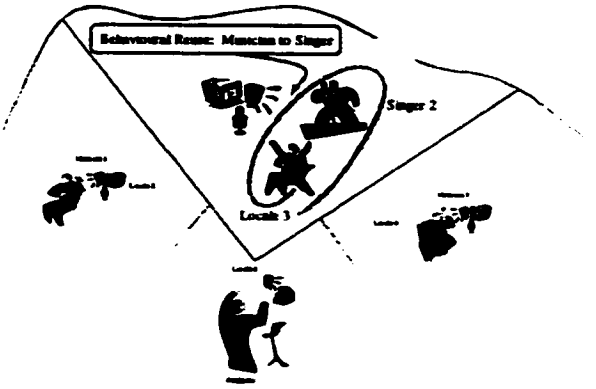


Figure 4.25: Behavioural Re-use in Tele-Orchestra Scenario

score in a manner other than the original. Because a jam session is typified by improvisation, this ability would effectively offer an "adaptive rehearsal hall" which would be able to modify its resource allocations based on typical behaviours of its participants.

4.2.3.2 Virtual Theatre

In an artistic venue like that of a theatre, a myriad of possible interactions can occur between various entities, ranging from the director and actors to the audience as well as the props and stage equipment. Typically, however, actors collaborate together under the auspices of a director with a preset story line to produce a dramatic production. In doing so, they can use props, see and talk to each other, hear and see prompting from the director as well as be aware of their environment, including (to some degree) the audience. Therefore, the actors and the director need to coordinate their actions in terms of order and timeliness, manage the use of props and other parts of the theatrical environment as well as have an awareness of the overall state of the production so they can adapt themselves to (i.e. *ad lib*) the developing story line.

When attempting to provide for a virtual theatre, in which actors, director and audience are located in different physical areas, connected to a distributed environment that functions as the theatre, the same challenges exist. That is, the actors and directors much be aware of and able to interact with each other while the audience must be aware of the actors on the stage as they are performing. Manipulation of props and restrictions of the environment must also be observed (just as a real actor shouldn't fall off a stage, neither should a virtual one be able to perform unreasonable or undesirable actions). In many senses, the virtual theatre scenario shares many of the same characteristics and requirements of the tele-orchestra: the ability to hear dialogue and/or audible cues is fundamental to engaging in an aural interaction between the those in the space. Additionally, the need to "see" what is going on in the shared space (both for the actors and the director) is required for their analysis and continuing collaboration. The audience also has a need to hear and see the actors but unlike the actors, the audience should not be aware of the director's interaction during the performance. Therefore, the ability to be on/off-stage is still necessary.

The novelty of the virtual theatre concept implicitly implies a variety of possible user interfaces. One example includes the use of avatars which would act as placeholders for actors within the environment. In such a system, a person could interact with another via manipulation of their respective avatars (in a manner similar the user interface found in systems like [ABH+95] and [MBD+95]). Alternatively, real-time audio/video capture and display could be used to provide a "reality-based" interface via a television/teleconferencing-like paradigm. Or, the two techniques could be combined to provide real-time video/audio supplements within an avatar-based interface. Figure 4.26 illustrates an avatar-based user interface in which actors are shown on a shared stage that can be seen via a remote audience. The director is also shown as an off-stage avatar – one that can be seen by the actors on stage but not by the audience. As discussed shortly, this concept of stage presence and view position can be facilitated using the appropriate collaborative associations and their membership policy (i.e. CAMP) to limit participation in collaborative groupings and the effects of collaborative focus.

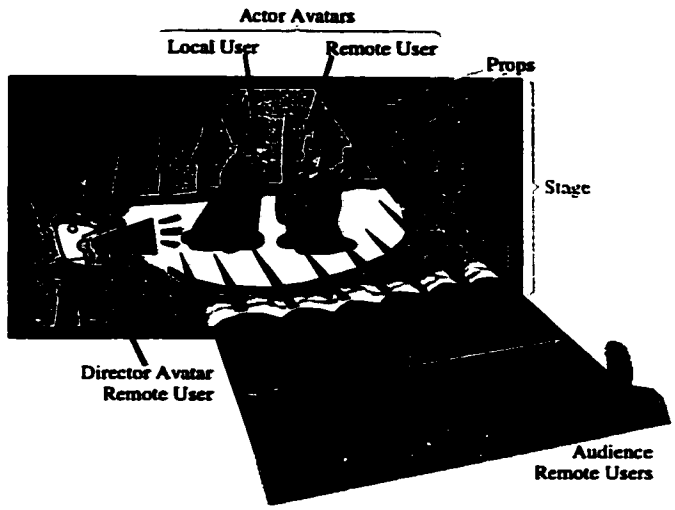


Figure 4.26: Virtual Theatre Scenario with Conceptual Interface

This independence of the user interface relative to the scenario and entities concerned illustrates the potential of separating and addressing concerns in an orthogonal manner. By providing user interface and event coordination at a meta level, some users could utilize a 2D interface while those with the appropriate facilities could use a much more elaborate UI metaphor (such as 3D immersive VR). Even though the UI would be different, the coordination logic would still be the same and not dependent on the collaborative object's representation per se, but rather a lower-level control pattern that maps specific events to specific aspects of a particular UI technology. Consequently, global changes to the actor roles, their interaction and representation could be done independently.

For example, consider the virtual theatre scenario outlined in Figure 4.27 which represents the scenario analogous to that in Figure 4.26. The CSIG/CAAD pair outlined in Figure 4.28 and Figure 4.29 show the use of audio/video for the different classes of entities: that for actors and that for the director. As the audience is to

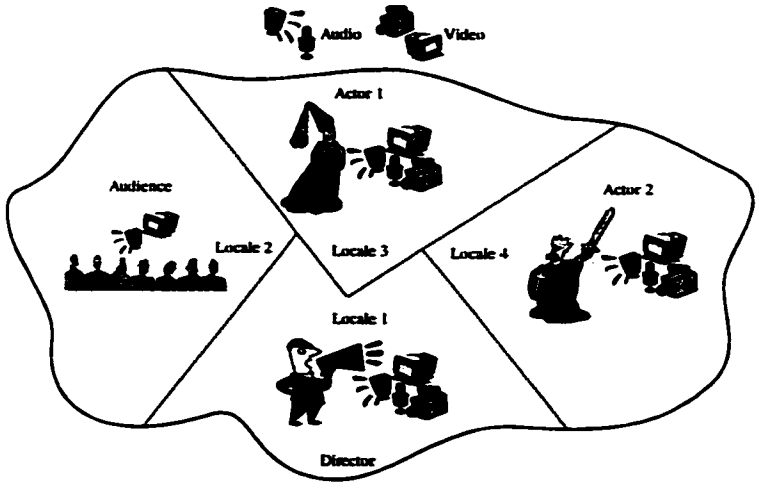


Figure 4.27: Conceptual Virtual Theatre Scenario

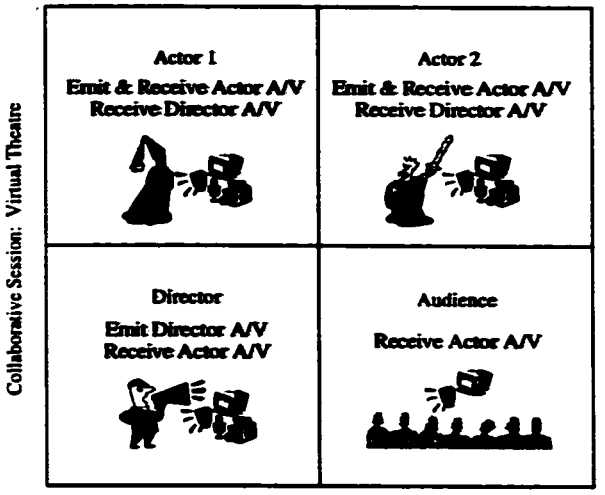


Figure 4.28: Virtual Theatre CSIG

hear/see the actors but not the director, by separating the audio/video groups based on actor/non-actor presence, it is easy to address audience awareness by their participation in the given collaborative group. Additionally, because the organization of the scenario is such that actor A/V is heard/seen by everyone involved, a reasonable design choice would be to use the default Null CA (●) to distribute the actors' audio/video tracks. Other less general activities, such as "emitting" and "receiving" audio/video from the director, are assigned to task-specific CAs (② and ③ respectively in this example). Specifically, this use of the separate "emit only" and "receive only" CA pairs for the director's audio/video suit the uni-directional nature of the control flow from director to actor. Conversely, the use of the Null CA can allow for universally bi-directional audio flow or just a limited implementation of the general pattern (i.e. receive only without emission) for the audience and director in terms of "receiving actor A/V". This flexibility promotes re-use of a very common pattern while allowing the potential for adding audio feedback from the audience (i.e. waiting for applause or laughter to quiet down and/or possible interaction with the audience for an interactive theatrical performance).

The corresponding collaborative behavioural tree shown in the left pane of Figure 4.30 shows one possible definition of the virtual theatre scenario in which all entities (the actors and the director) are directly controlled

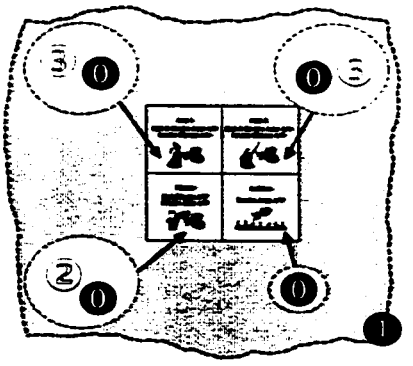


Figure 4.29: Virtual Theatre CAAD

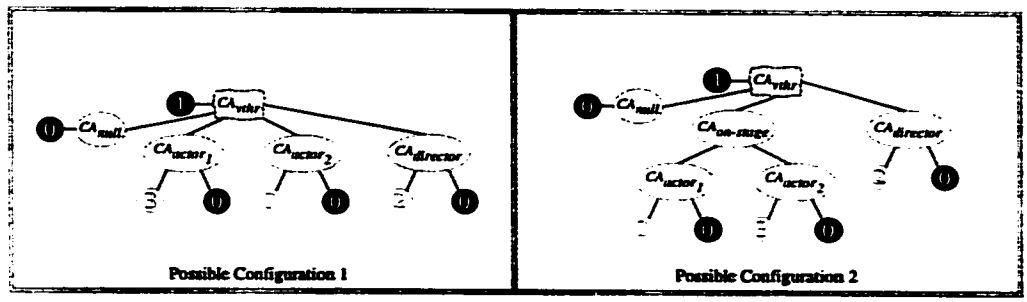


Figure 4.30: Virtual Theatre CBT Variants

in terms of the theatre scenario. Consequently, the theatre scenario directly controls the audience, the director as well as each actor. As with the above, the exact realization of the behavioural components is based on a composite CA which coordinates the semantic singleton patterns relative to the specific entity (e.g. for a specific actor). In such a CBT, stage presence, default (audience) point-of-view and the behaviour of each theatrical entity (i.e. actor) is controlled at a global level. While straight-forward, the scalability of such an approach is reduced because the 'theatre' itself must be adapted to the number of actors. The use of an "on-stage" CA (right pane, Figure 4.30) promotes scalability in terms of actors such as the director, audience and the "those on stage" interact while the "on-stage" CA deals with managing the specific interactions of those on stage vs. those off-stage (which could be given their own collaborative association if desired).

4.2.3.3 Tele-Dance

Another novel application of media space technology in the artistic arena is the tele-dance. In such a scenario, distributed participants dance together as if they were co-located with their partner(s). While such an approach begs numerous questions in terms of technological support, most expressly in terms of suitable user interface design, it is a good example of the need to apply both coordination, synchronization and the use of well-known and specific behavioural patterns. As an example of a more esoteric scenario, an advanced user environment like that in Figure 4.31 will be considered based on the technology outlined in Table 4.5.

The example uses force-feedback sensored body suits worn by the dancers to generate information (i.e. events) describing bodily motion and pressure exerted by physical gestures during the dance. These events, once transmitted between participants, could allow partners to "feel" what their partner is doing in terms of dance movements; that is, the force feedback sensors could simulate the pressure applied by a real partner in the dance to provide the sensory feedback without the actual participant being co-located. Alternatively, video composition and avatar technologies could be used (as illustrated in [MBD+95]) to manipulate graphical representations of the dancer's remote partner(s).

Various artistic opportunities can be envisioned for such a scenario as well as its application as a learning tool for people trying to learn particular dance steps. By matching a person's actual movements to the correct dance pattern, sensory feedback could be generated by the system to correct movements much like an actual dance instructor would do. That is, events generated by the dancers' movements can be coordinated relative to a well-

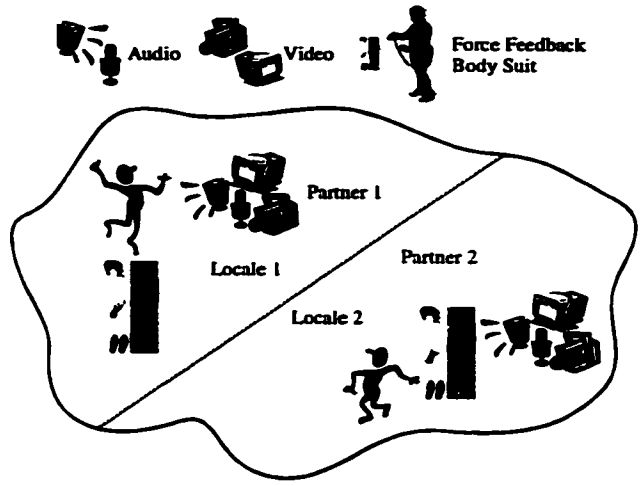


Figure 4.31: Conceptual Tele-Dance Scenario

Technology	Purpose/Facilitation
Audio capture and rendering	Talk/hear dance partner for instructions, music, aural cues
Video capture and rendering	Visualize dance partner, their behaviour and visual cues
Force-feedback body suits	Receive physically gestured cues from their partner(s) so as to mimic co-located physical interaction

Table 4.5: Tele-Dance Scenario User Interface Technology

known dance pattern (to facilitate correct performance) and each other (to match movements). A simple and recognizable example is the basic "waltz" pattern, whose steps and (possible) representation as a finite state machine are shown in Figure 4.32. Using such an executable model to represent dance steps allows a straight-forward correspondence between a dancer's movements (i.e. input events generated by bodysuit sensors) and their partner's interaction (i.e. computed responses realized by bodysuit force-feedback functionality). Subsequently, changing the dance that participants are trying to learn/perform (and hence have their movements compared to) would simply involve changing the dance pattern (i.e. its finite state machine) with the appropriate definition, so movement events are compared and recognized by the appropriate FSM.

The basic two-person tele-dance scenario is shown in the usual diagrammatic fashion in Figure 4.33, Figure 4.34 and Figure 4.35. The given example uses three CAs to facilitate the media-level interaction between participants; their rationale is offered in Table 4.6.

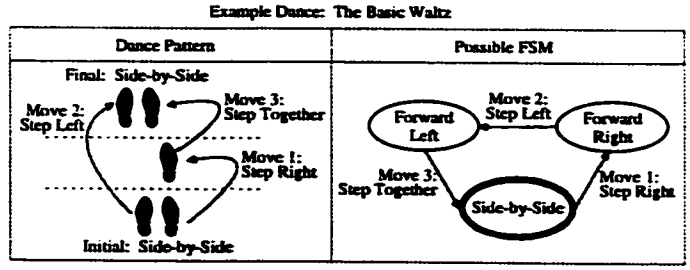


Figure 4.32: Example Tele-Dance Pattern and Possible Executable Representation

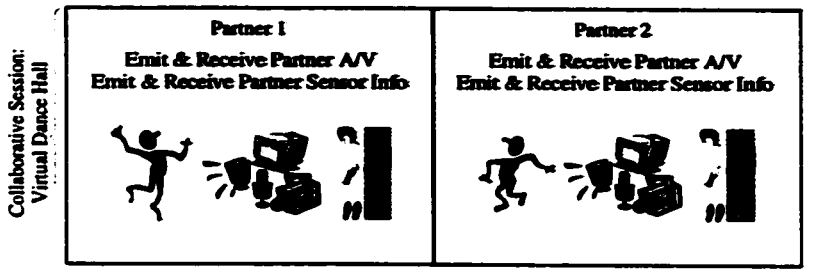


Figure 4.33: Basic Tele-Dance CSIG

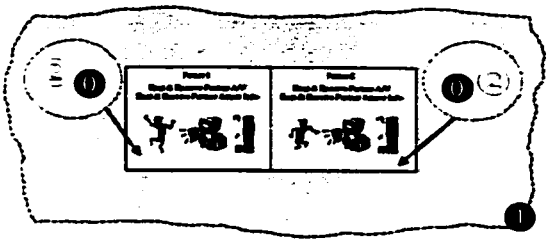


Figure 4.34: Basic Tele-Dance CAAD

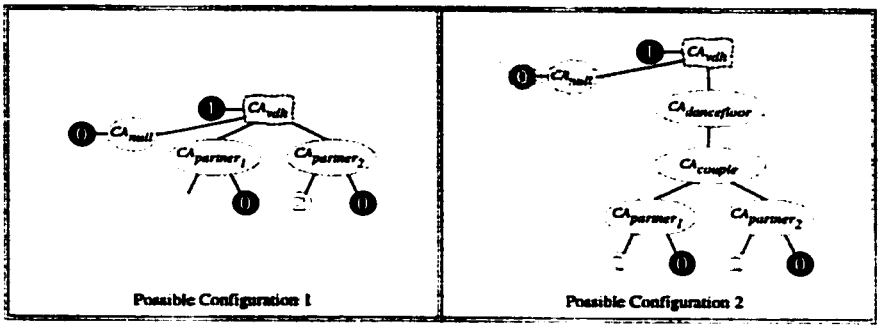


Figure 4.35: Basic Tele-Dance CBT Variants

CA	Domain	Rationale
②	Sensor	Body movements shared only between participants
●	Dance Scenario	Defacto scenario definition
●	A/V	Audio/video of dancers; default CA is amenable to general distribution for all participants including an instructor or receive-only audience

Table 4.6: Basic Tele-Dance CA Distribution, Domain and Rationale

Despite the use of only two semantic singletons (albeit complex ones in this illustration), Figure 4.35 shows two possible CBT organizations: (1) dancers are directly subject to the management of the main dance scenario (left pane); and (2) controlled at a finer and individual granularity using a "couple" CA (right pane). The second option promotes scalability in terms of coordinating additional dancers and offers the ability to more easily provide different rules to manage each pair individually. The right pane also shows the use of a "dance floor" CA which is akin to being on- and off-stage in the virtual theatre example. By doing so, support for a viewable dance floor along with a passive audience and off-floor instruction is more easily facilitated.

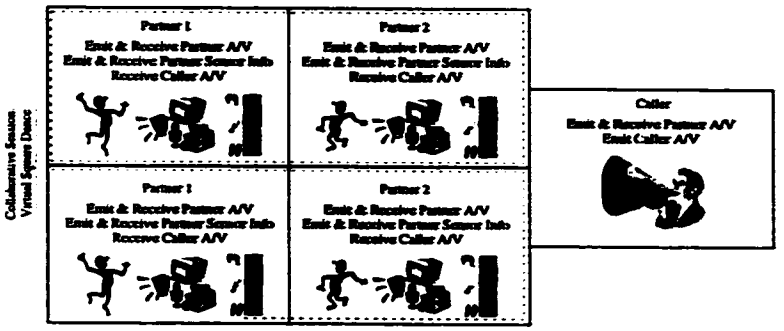


Figure 4.36: Advanced Distributed Square Dance Scenario CSIG

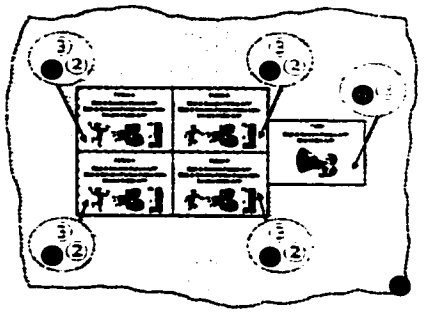


Figure 4.37: Advanced Distributed Square Dance CAAD

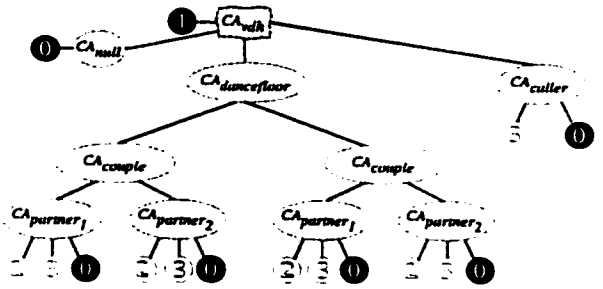


Figure 4.38: Advanced Distributed Square Dance CBT

The basic tele-dance example could therefore be extended as in Figure 4.36, Figure 4.37 and Figure 4.38 to show support for a distributed square-dance scenario with an off-floor "caller". While support for the different dance steps would be provided by a different dance pattern, an additional CA (number ③) is used to represent the caller and provide audio direction to the existing partner groups. Additionally, use of the Null CA for dancer A/V automatically supports the caller being able to see and hear what the dancers are doing without the introduction of an additional CA for an already existing behaviour.

4.2.4 E-Commerce

As part of the increasing permeation of network technology into homes and businesses, electronic commerce continues to emerge as an increasingly popular alternative to traditional commercial activity. Consequently, the importance of facilitating the diversity and flexibility required by such systems is an important issue.

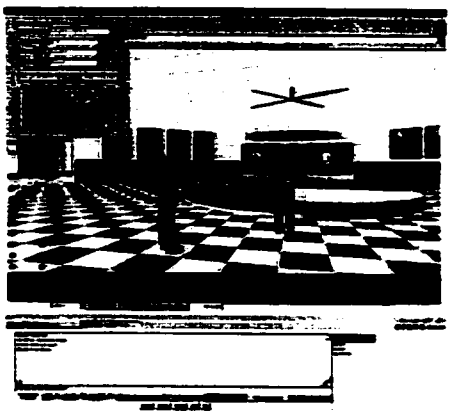


Figure 4.39: Example E-Commerce Interface [OMS+00]

One often cited example is that of an "e-tailor"; that is, a scenario that enables electronic retailing (i.e. shopping at retail outlets). The versatility of a media space enables many possible configurations which could be useful in such a context. For example, a user interface (such as in Figure 4.39) could provide an avatar-based virtual world at which users move throughout a three dimensional representation of the commercial environment. Alternatively, a standard web browser-based catalog interface could also be provided for users with less powerful equipment. Regardless, the opportunity to interact with sales staff, examine as well as purchase products all require some mechanism to govern the interaction between the staff and customers, the logic of the business transactions and the manipulation of objects and other media used in the sales/demonstration process. Again, because the user interface is defined at the meta-space level, a system could be designed for multiple UIs based on the same commercial transaction logic. Consequently, a family of "e-tailor" products could be created to offer scalability and diversity in terms of transactional behaviour but also its user interfaces.

As an example, Figure 4.40 illustrates a conceptual shopping scenario consisting of a migrating shopper who moves between areas in a virtual clothing store as part of a virtual shopping mall. Divided into interaction groups in Figure 4.41, a possible CA allocation is shown in Figure 4.42; its rationale is given in Table 4.7. As with the earlier examples, the use of a migrating member allows a user to engage themselves in different kinds

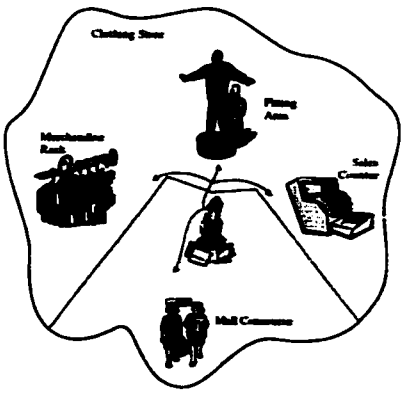


Figure 4.40: Conceptual E-Commerce Scenario

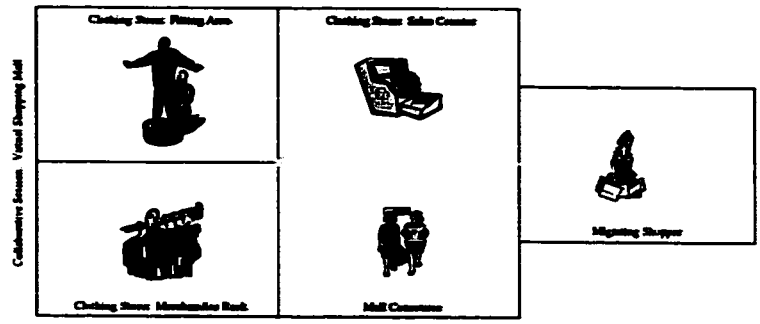


Figure 4.41: E-Commerce CSIG

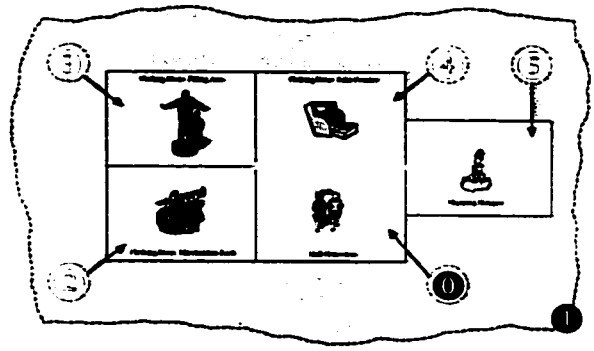


Figure 4.42: E-Commerce Scenario CAAD

of interactive behaviours. Based on the use of the CA membership policies (i.e. CAMPs), different boundaries for interaction within the shopping environment's logical space can be used to tell whether or not the shopper is browsing, "trying on" or purchasing merchandise. By providing different CAs for the different "parts of the store", different aspects of interaction can be addressed; for example:

- Varying amounts of inter-personal scope can be offered – hearing/seeing other people in a 3D version of the merchandise rack area may be desired to create a more realistic shopping environment while only being aware of the sales person may be all that is desired during the sales transaction itself.
- Varying levels of security may need to be associated within different types of behaviour; for example, the

CA	Domain		Rationale	
⑤	Migrating Shopper		Default "private" behavioural environment when not interacting	
④	Clothing Store	Sales Counter	Individual Shopping Context	Sub-area dealing with sales behaviour (e.g. isolate security concerns)
③		Fitting Area		Sub-area dealing with "trying on"/viewing merchandise
②		Merchandise Rack		Sub-area dealing with displaying merchandise
●	E-Tailor Scenario		Defacto scenario definition	
●	Mall Concourse		Default "public" behavioural environment shared amongst all members of the scenario (i.e. akin to walking or sitting in the open mall concourse and experiencing all the other shoppers)	

Table 4.7: E-Commerce CA Distribution, Domain and Rationale

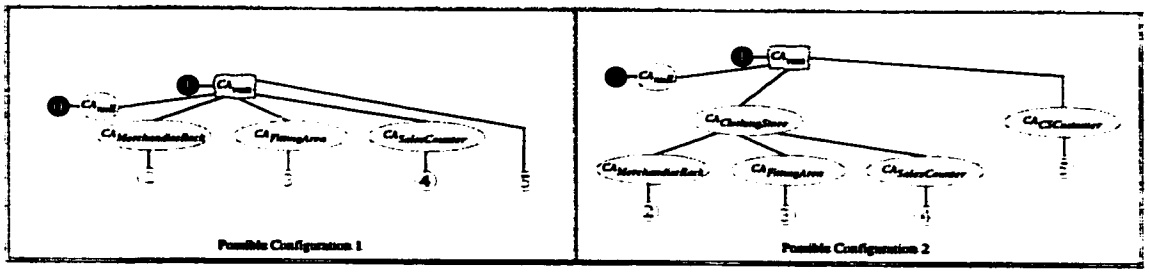


Figure 4.43: E-Commerce CBT Variants

sales counter CA would need more security to protect financial information (such as credit card information) when a purchase is being made.

- Different user interfaces may be desirable for the different areas of the store; for example, a frontal/side-view browser with article movement capabilities (i.e. clothing rack 'scroll bars') could be useful for the clothing rack CA. In the fitting area, multiple perspectives based on a three-dimensional panoramic view of the product (such as on a mannequin) could be useful. The sales counter, however, would doubtfully benefit from the complexity associated with these kinds of interfaces. In short, each store area is better served by the ability to isolate its particular UI local to its own CA.

As shown in Figure 4.43, different collaborative behaviour trees are possible depending on the organization and delegation of management responsibilities within the scenario. The left pane shows the simplest configuration in which the virtual shopping mall directly manages the migrating shopper and the different areas of the store. While simple for a small e-commerce environment, expansion of the environment both in terms of size and diversity would complicate the main scenario definition. Using the right pane, in which a store manages its own local subdivisions and customers are dealt with as their own particular behavioural category offers a more flexible and scalable approach.

4.2.5 Multimedia Synchronization

The preceding example scenarios have shown the application of RASCAL as a means to facilitate flexible and manageable collaboration. In these examples, the use of different multimedia tools have been assumed as part of the available environment without much consideration for how they are handled within it. This section now explores the application of the RASCAL framework to this level of collaborative multimedia: how RASCAL facilitates managing individual and grouped multimedia constructs.

As mentioned earlier in Chapter Two, numerous approaches to multimedia synchronization can be found in the literature. Of these, many deal exclusively with synthetic media while some handle conversational media like real-time audio. To meet the diverse needs of collaborative practices, however, collaborative multimedia must handle both synthetic and real-time media simultaneously and in conjunction with each other. For example, in addition to supporting synchronized playback of a pre-recorded audio and video tracks, a collaborative system may require the ability to synchronize a live audio commentary to the pre-recorded video. Such a scenario is an

example of *Mixed Mode Synchrony* – the ability to combine, synchronize and coordinate media of different modalities (i.e. both real-time and synthetic) within the same environment.

Like the process of collaboration itself, the behaviour of real-time media is open-ended; the contents, semantics and behaviours of "on-the-fly" media are indeterminate and not well-known in advance. Synthetic media, on the other hand, have known characteristics which can be analyzed prior to use. Rather than reconcile these divergent viewpoints, systems often limit their functionality and support for these differing paradigms. The result is system rigidity: provision for specific interactive behaviours, specific user interfaces for their representation and specific mechanisms to implement them. In short, they are not open-ended and they are not adaptive. Presentational systems, for example, typically utilize fixed playback schedules with ordering and inter-media dependencies determined/computed before playback. While suitable in certain instances, such an approach does not offer dynamism, flexibility or fluidity in interaction and usage. Conversely, while conversational systems often utilize a protocol-based approach to be more reactive to the real-time data flow characteristics, the same techniques are not necessarily appropriate for presentational systems. In particular, because presentational data flows are well-defined and can be characterized in advance, advance buffering and other optimization techniques can be applied ensure timely data availability. To cover this breadth of utility, systems can put different kinds of functionality together in a large software package. The result is a collection of narrowly-scoped techniques that makes for increased complexity due to inclusion without the flexibility of integration. Such growth in functionality does not necessarily mirror itself directly in terms of flexibility, but rather in the volume of choice. Furthermore, unlike functionality, if the system is not designed to offer the desired behavioural support, there is typically no way to provide it. Therefore, if such behaviours are required, they must be ensured manually (Figure 4.44).

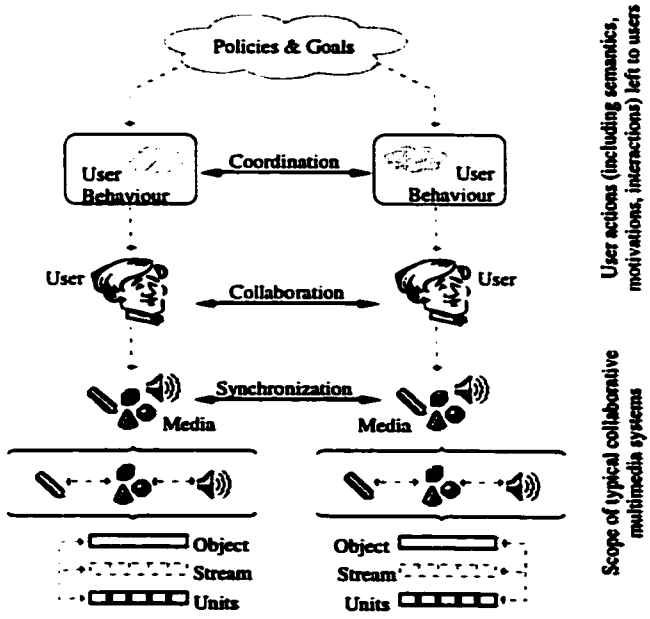


Figure 4.44: The Usual Divide – Collaboration, Coordination, Synchronization

This section illustrates the orthogonality of RASCAL by showing its application to common multimedia synchronization scenarios. In particular, two important categories of multimedia synchronization are overviewed: (1) audio/video synchronization; and (2) mixed-mode synchrony.

4.2.5.1 Audio/Video Synchronization

The classic audio/video synchronization scenario is often called the "lip sync" problem, and as presented previously, it has been extensively studied in the literature. Typically, it is addressed in two separate contexts: (1) synthetic pre-recorded A/V streams; and (2) live real-time A/V streams. In each case, however, the goal is the same – to align audio and video streams within a given tolerance (Figure 4.45).

While the previous applications of the RASCAL framework have illustrated somewhat high-level behaviour, the notion of audio/video synchronization (i.e. aligning correct portions of audio/video streams) is in itself a behaviour. Consequently, the same techniques can be applied at a lower-level of abstraction, therefore offering an orthogonal approach to behavioural management. Because the exact mechanism and policy logic are defined at meta-levels, the same framework can be applied since the aspects which must change to address issues of complexity vs. implementational efficiency are already separated from the organizational structure.

As an example, consider Figure 4.46 which illustrates RASCAL's meta-object separation of concerns for the audio/video synchronization scenario. A possible use of the class framework introduced in Chapter Three is shown in italics. In this example, the inter-stream aspects of policy and coordination are shown as meta to the intra-stream mechanisms and the policies that directly manage them. This perspective matches the premise that in order to achieve inter-stream synchronization, intra-stream synchronization must be achieved first. While the concentration of the example is the synchronization and coordination of the streams at the rendering level, the intra-stream coordinator could also be connected to the policy that manages the data flow for each object (via the appropriate Data Flow Policy). By doing so, rather than strictly making changes to the final rendering of a medium, a coordinator could regulate data arrival (such as dealing with QoS issues via the object's meta space).

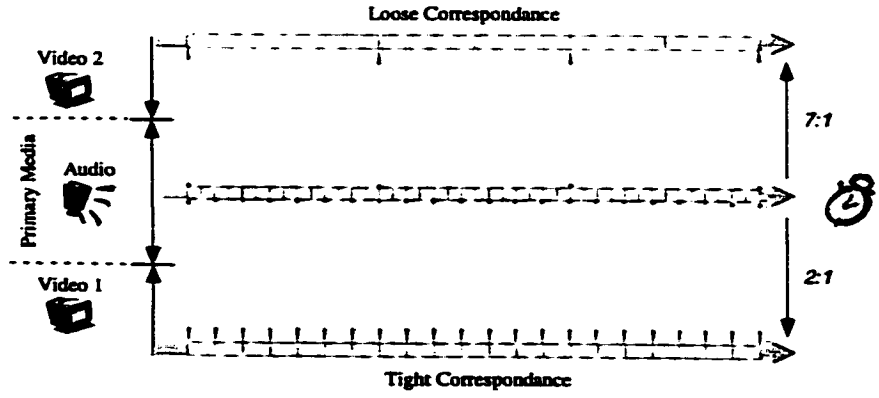


Figure 4.45: Example A/V Temporal Alignment

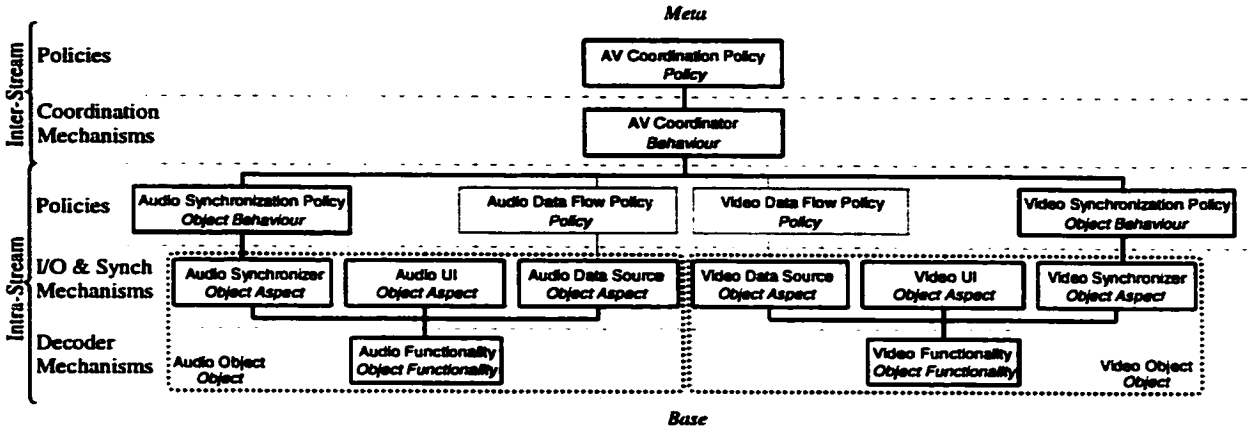


Figure 4.46: AV Pattern Organization

Another benefit of the meta component-based approach is the versatility brought about through customization and re-use. As an example, Figure 4.47 shows an A/V semantic singleton in which two different entities are replaceable: one structural and one behavioural. The rationale behind these customizations is given in Table 4.8. Consequently, the same basic construction can be seen to offer (in the particular configuration shown here) up to four different permutations. By localizing such variations to a well-known pattern, the pattern can be used as a general solution step by more abstract models. For example, the A/V pattern can be used as a component by more complex scenarios without them being concerned about the details of audio/video synchronization. However, because the solution uses an orthogonal approach to the more abstract collaborative scenario descriptions, the same methodology/management and support infrastructure can serve both purposes.

4.2.5.2 Mixed-Mode Synchrony: Beyond Structural Composition

While audio and video constitute an important part of collaborative multimedia, broader support for multimedia is key to its utility. Such support is commonly addressed by adding different functionality through the use of

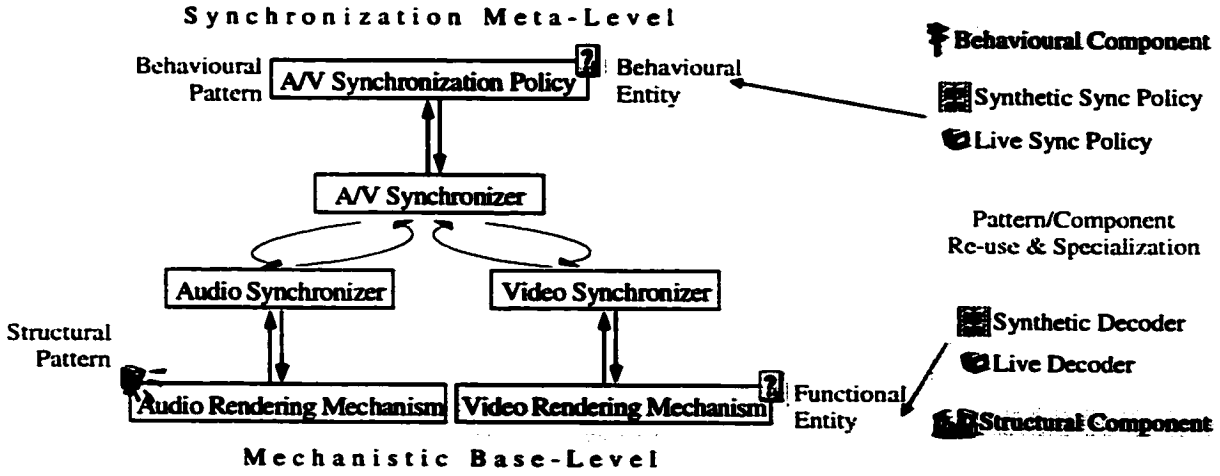


Figure 4.47: AV Pattern Specialization

Entity Type	Entity Purpose		Entity Difference	Rationale
Behavioural component	Video	synchronization policy	live vs. synthetic	Deal with asynchrony (hold frame, go blank, ...) or change frame rates differently/accordingly
Structural component		decoder mechanism		Can use more computationally light-weight decoder for real-time media

Table 4.8: Multimedia Component Specialization – Purpose and Rationale

"plug-ins", dynamic link libraries or similar technologies. However, the result is typically an array of functional (i.e. structural) capabilities, independent of support for how they work together. In the context of collaboration, this functional diversity without support for behavioural integration and management results in an unwieldy environment that lacks support for cohesion and coherency.

The prime difficulty with the open approach, however, is how to deal with all the possibilities that can occur. For example, if only two specific types of media are used, the system can be built to suit their particular characteristics. However, in a system in which "everything can change", static structural and behavioural models (including composition, dependency and synchronization rules) are not suitable. For example, the above case could use static behavioural rules defined when the system is built or the scenario authored; e.g. "video 1 has audio annotation 2 for time 20" with precedence "annotation masters video" would synchronize video relative to the audio annotation. While useful for synthetic "view only" presentations, modifications or extensions to the system could be awkward, difficult and expensive. Consequently, in a system which is typified by the necessity to change, such an approach limits collaborative potential.

While an A/V scenario epitomizes the notion of temporal synchronization as applied to continuous media (i.e. rendering the media streams according to timeliness of playback), collaborative multimedia requires broader support for different media types as well as different synchronization techniques (see Chapter Two and Appendix A). For example, collaboration that utilizes media, applications and related tools ranging from streaming video to structured drawing applications must consider not only temporal synchronization, but the effects of causality between interacting objects. Doing so, however, is complicated because mixed-modality involves support for real-time and non-real-time media. This mix of the inherently unpredictable and the predictable, combined with issues such as different synchronization granularities, precision requirements and so forth, clearly illustrate the complex and composite behavioural nature of such an environment. In short, the potentially heterogeneous mix of activities results in a "multisynchronous – multimodal" space in which the infrastructure relating to object interaction must be flexible enough to handle variety and manageable enough to control it.

An example of temporal vs. causal linkage between media is illustrated in Figure 4.48 in which the progression of a slide show is controlled in one of two ways: (1) temporal, in which the display of each slide is timed; and (2) causal, in which progression between slides is based on the occurrence of a specific event (in this case playing the high A and low G notes). As such, the event-driven slide show illustrates the notion of mixed media casual linkage using semantic events which act as triggers. Mechanisms for how to realize such trigger events in a continuous real-time music stream, while important, are a meta-level issue and would be part of the

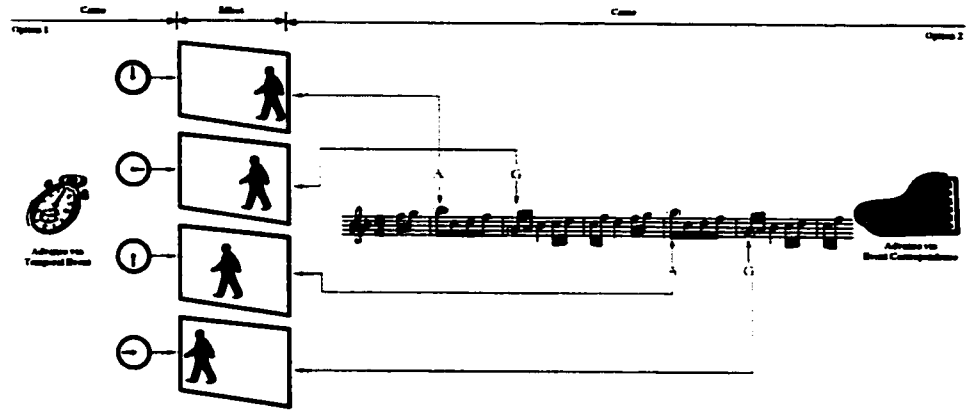


Figure 4.48: Temporal vs. Event Correspondence Between Media

appropriate collaborative object's meta space. The actual relationship for the specific types of triggers and the synchronizable relationship to other media would be the responsibility of the related collaborative association.

A similar scenario based on the interaction of three different kinds of media is illustrated in Figure 4.49. Unlike the previous example, this tri-entity relationship uses a voice/keyword recognition interface to trigger the advance of the slide show. Additionally, the correspondence between slide and film frame advance would be specified as part of the collaborative association managing the trio's interactions. In this case, the interface for each object, their functional logic, their individual behaviour and their joint behaviour are addressed separately. As per the RASCAL framework, these aspects map to the appropriate meta spaces and their realization would be facilitated using active objects which independently compute and perform the necessary role logic. That is, the media actively compute about their own role in the synchronization process.

Another example, based on the notion of a television news broadcast, shows the application of relationship between multiple continuous media streams at different levels of granularity/precision. Shown in Figure 4.50, the news anchor's commentary provides the primary audio stream which masters the two video streams. The

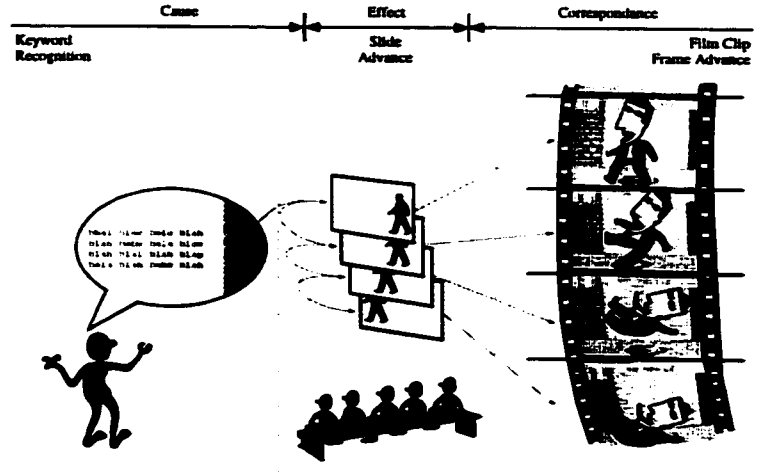


Figure 4.49: Mixed Media Scenario with Event Correspondence

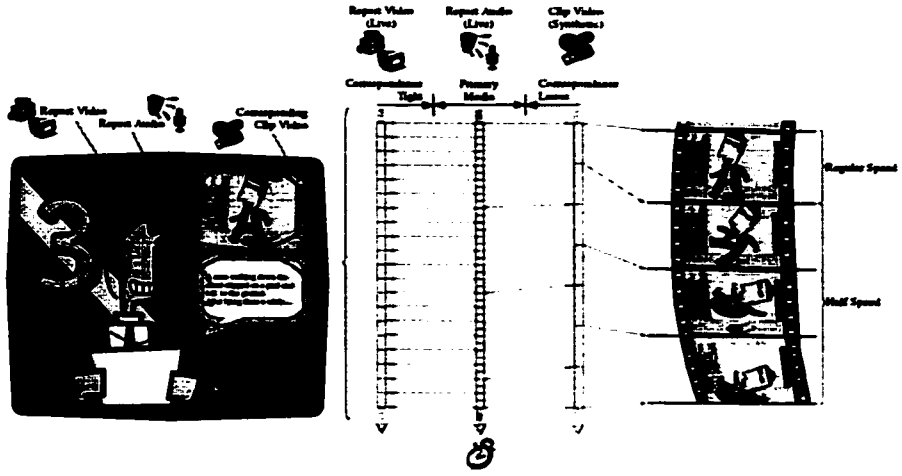


Figure 4.50: Multi-Granularity Mixed AV Scenario

first video stream, labelled as report video, is the video track that corresponds to the audio commentary and requires tight synchronization to ensure "lip sync". The second video stream (labelled as clip video) is related to the other two streams in that it is being described by them; that is, it has a holistic relationship at a larger granularity to the two other streams which have a tighter semantic correspondence. Therefore, a possible solution to such a situation could be to configure the synchronization policies that manage the two audio/video relationships to have a "collaborative semantics" meta object in the policies' meta spaces (see Figure 4.51). This semantic object could specify (in some system-dependent manner) the appropriate priority between the streams (e.g. audio is master, tight vs. loose semantics for A/V streams). It could also indicate which media to alter so they "fit together", such as changing playback speeds (Figure 4.50).

While the preceding multimedia examples generally concentrate on "low-level" aspects of synchronization, the question remains how to use the framework to synchronize and coordinate media in a more abstract and high-level manner. For this situation, the mixed media scenario illustrated in Figure 4.52 is used to show the interaction between a shared document edit, a slide show and a film (video) clip. While these individual tasks can be well-handled individually, consider a dependence between these groupings as shown in Figure 4.53. The dependency is that a piece of text in the textual document is semantically linked to the third slide in the

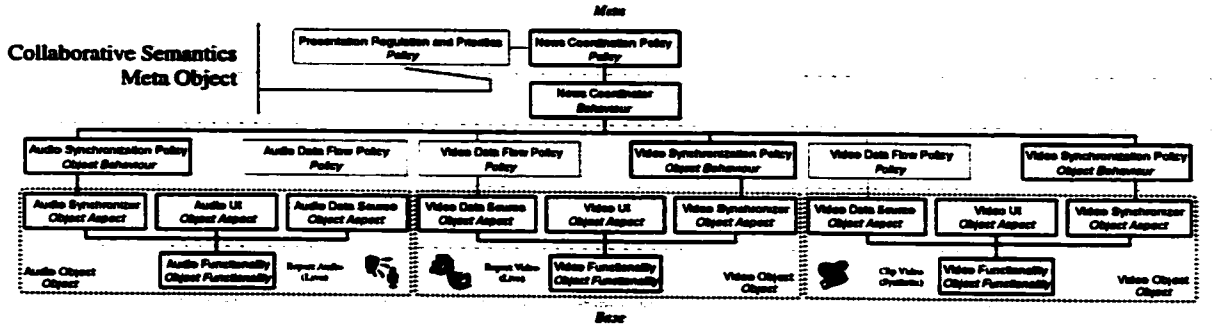


Figure 4.51: Possible Organization of Meta Objects for Mixed AV Scenario

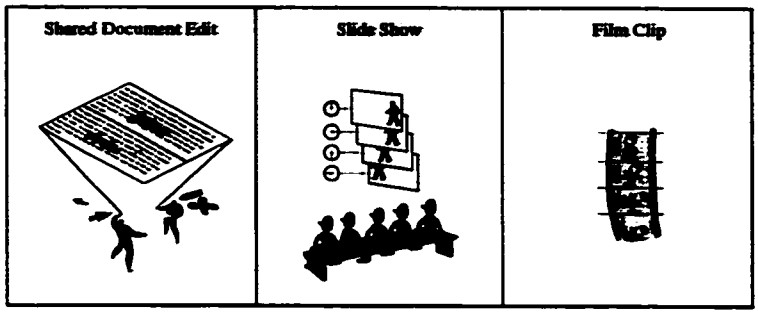


Figure 4.52: Conceptual Mixed Media CSIG

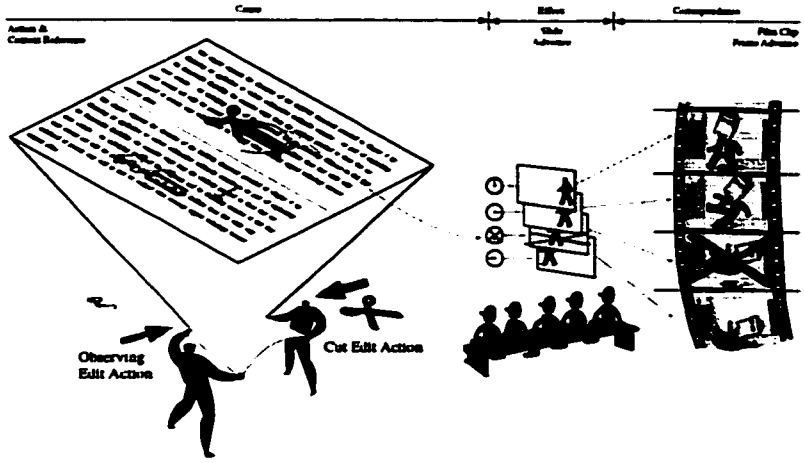


Figure 4.53: Example Causal Dependency Between Mixed Media

slide presentation, and transitively, the third frame of the film clip (as illustrated both here and earlier). Therefore, a "causal ripple through" as illustrated in Figure 4.54 would occur amongst the various facets of the scenario when the said text was deleted (shown as being "cut" from the document).

Specification of these relationships is done in the usual manner using collaborative associations to group the specific behavioural logic. Shown in Figure 4.55, Figure 4.56 and Table 4.9, this set of activities could constitute its own scenario (left frame) or be a part of a larger collaboration (right frame with additional

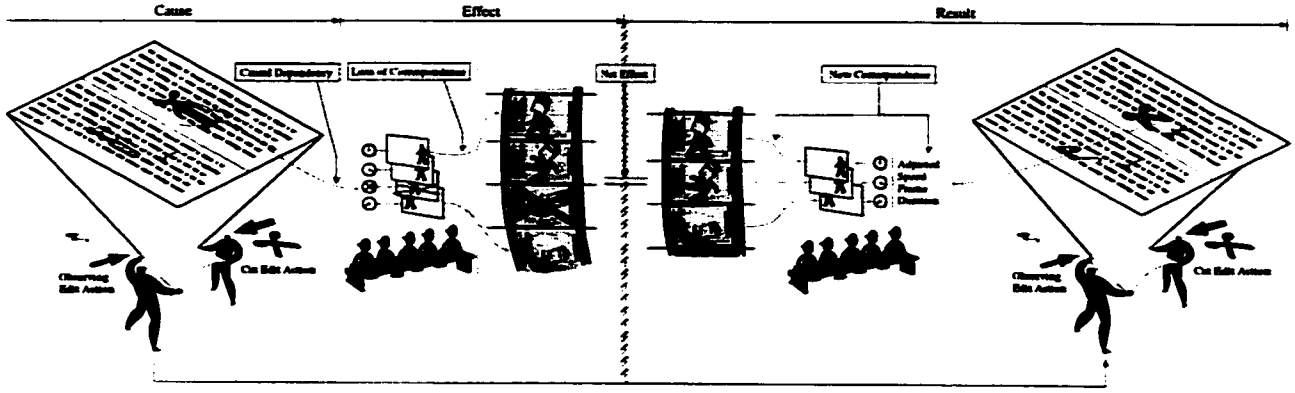


Figure 4.54: Example Causal Ripple-Through Between Mixed Media

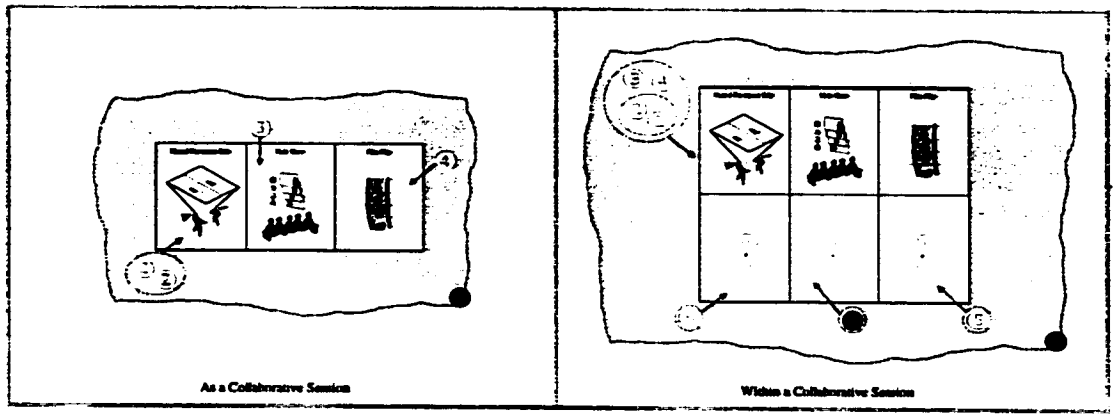


Figure 4.55: Mixed Media Scenario CAAD Variants

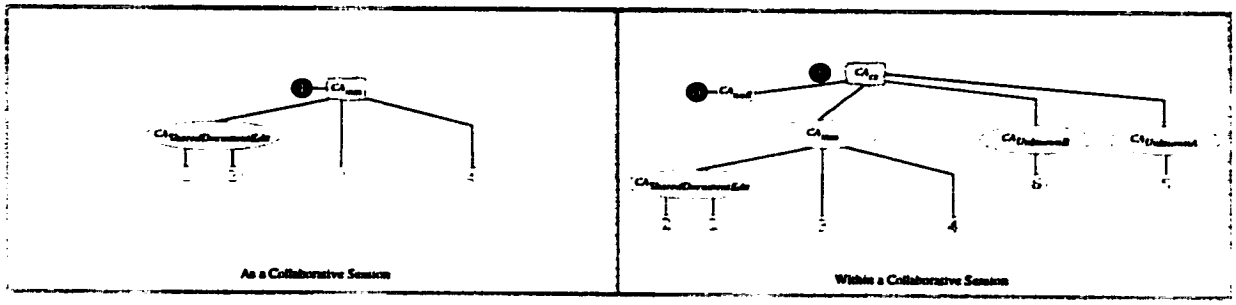


Figure 4.56: Mixed Media Scenario CBT Variants

unspecified CAs). Specifically, CA_{mm} is shown as the CA that manages the relationship between the various individual media (the document, the slides, the film). In the left pane, it is shown as the root CA while in the right pane, it is shown as a composite that manages the multimedia parts of some larger scenario. The unspecified composite CAs ($CA_{UnknownA}$, $CA_{UnknownB}$) as well as the Null CA and those labelled ⑤ and above are simply placeholders meant to illustrate that more scenario complexity could be derived in a similar manner. Unlike previous examples, the Null CA is not used for an audience broadcast because the intention of the scenario is that an audience would be relative to a specific medium (for example, certain people would see the slides but not be aware of the document). Therefore, since ③ is already serves as a "passive read-only" slide presentation group, joining that CA would be to experience the slide show.

The shared document edit collaborative association $CA_{SharedDocumentEdit}$ is shown as a composite behaviour which deals specifically with joint document editing; that is, it regulates multiple concurrent instances of the basic

CA	Domain	Rationale
④	Video Playback	Specifies film frame playback/timing behaviour
③	Slide Show	Specifies slide presentation/timing behaviour
②	Single User Edit	Specifies edit logic for a single user
●	Media Scenario	Defacto scenario definition
●	Not Specified	Not specified in this example
⑤+	Not Specified	Not specified in this example

Table 4.9: Mixed Mode Multimedia Example CA Distribution, Domain and Rationale

editing CA which is shown as semantic singleton ②. Such a construction allows joint editing to be managed in a scalable manner independently of the basic edit process, since management of the group is performed at a meta level to the actual edit logic. Overall, however, these behaviours would specify how the (joint) editing is performed (i.e. locking, optimistic versioning and such) while the CA that relates the editing process to the other media (i.e. slide show and film) would be specified separately. That is, this dependency would be provided as its own CA, which through the use of the active object paradigm, would allow the relationship to actively evaluate whether the system was conforming properly. Subsequent to the edit (text deletion), the individual synchronization behaviours of the slide show and film clip could then perform different re-synchronization actions (specified locally to themselves). For example, Figure 4.54 shows the duration of the slide/frame display being adjusted to meet the new conditions. Whether doing so is a valid option, however, is dependent on logic specified in that CA's meta space (such as specified by the collaborative semantics meta object). Other actions, for example, could involve "holding" the current frame, "going to black" or any number of transitional behaviours.

The main principle in the above example, is that provision for the transition, the selection logic to apply the said transition and the management of the dependency are all handled separately. This separation is inherently more scalable since it combines straight-forward and general principles/policies rather introducing scalar (dimensional) restrictions into specific definitions. Additionally, through the use of the active object and reflective paradigms, each of these aspects can perform real-time monitoring and evaluation of their situation and modify themselves and/or related entities to meet the needs and realities of the current collaborative environment.

4.2.6 Multimedia Summary

Support for multimedia synchronization in a dynamically changeable environment necessitates the provision of a dynamically adaptive synchronization architecture. As applied to mixed-mode synchrony, this includes the ability to address the myriad of characteristics than influence individual media and how they work together. Additionally, the framework was shown to be able to not only address abstract collaborative groupings, but also provide an orthogonal approach to traditional multimedia synchronization issues such as event vs. continuous synchronization, precision and granularity requirements, paradigm and causal dependencies across media of varying types and modalities.

4.3 Summary

This chapter has illustrated the RASCAL framework as applied to a variety of collaborative multimedia scenarios. The framework was applied over a range of abstraction levels and its ability to scale between different levels of technical detail was shown. Additionally, the various alternatives possible within a single scenario demonstrates the framework's flexibility relative to a given collaborative context and the viewpoint of the system designer/architect.

Ultimately, the essence of RASCAL as applied to collaboration and multimedia is its ability to provide a dynamically adaptive behavioural management architecture that can change itself to meet the differences in a collaboration and its media over time. To meet these needs, the RASCAL framework offers an extremely flexible way of addressing varied and disjoint requirements. Using an abstractable, open-ended and orthogonal approach to behavioural provisioning and management, RASCAL promotes a wide-range of adaptivity through support for customization, re-use and evolution.

The next chapter will now illustrate one possible approach to implementing the basics of such a system.

5 CHAPTER FIVE

REALIZING RASCAL II: THE MSPACE CONCEPTUAL PROTOTYPE

This chapter augments the previous discussion of the RASCAL framework by presenting the essence of the conceptual prototype implementation undertaken as part of this research. As an abbreviation for "media space", MSpace blends simulation and implementation technologies to illustrate the various aspects of the RASCAL framework. As such, it does not provide an exhaustive or complete illustration of all possible approaches to or benefits of a RASCAL-based system; rather, it attempts to show the general organization and potential of the framework in terms of one possible implementation.

First, the overall organization of the MSpace prototype is presented along with the use of specific technologies and their rationale. Individual aspects of the prototype are then detailed in further detail while evaluation of their suitability relative to system realization is offered in subsequent chapters.

5.1 Technological Organization and Rationale

The original motivation behind the design of the MSpace prototype was to illustrate a real-world collaborative user interface utilizing an intelligent communications subsystem vis-à-vis a reflective, model-based management system. The intention was to provide a concrete collaborative multimedia experience which was transparently managed by a reflective behavioural backbone. Based on the active object/executable model paradigm, such an approach suited both prototyping and simulation needs for the study of different behavioural patterns as well as issues surrounding its reflective design.

Toward this end, MSpace employed a variety of technologies to implement the different aspects of a RASCAL media space. Availability, relative suitability and the potential to illustrate relevant concepts were all considered in the selection of which methodologies and tools were utilized for different components. The organization and role of these implementation technologies is illustrated in Figure 5.1. Briefly overviewed in Chapter Two, their deployment within MSpace can be seen to fall within two distinct categories: (1)

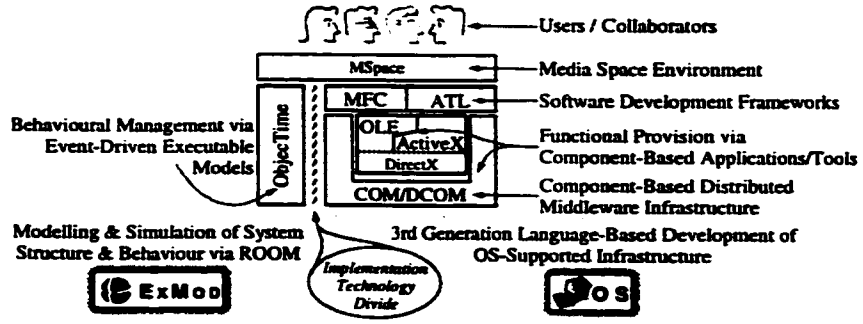


Figure 5.1: Prototype Implementation Technologies, Relationships and Roles

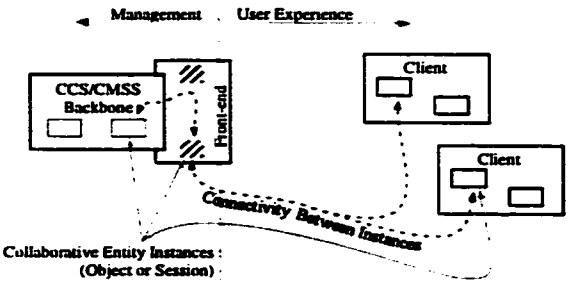


Figure 5.2: Collaborative Entity Correlation Across the Divide

behavioural management via modelling and simulation; and (2) functional provision and distribution via operating system supported infrastructure. In general, this division maps to a separation between system management and user experience (shown along with their correlation in Figure 5.2). Users interact via collaborative objects within the logical bounds of a collaborative session, both of which are provided to the user via a media space client that forms their "point-of-presence" within the space. Each client, session and object communicates via the CCS/CMSS which addresses their communication and behavioural management needs. In essence, the CCS/CMSS forms the "(behavioural) management backbone" while a front-end to the CCS (the CCSfe within the prototype implementation) provides the transition point between end-user and management technologies . The allocation of specific technologies to these architectural components is summarized in Table 5.1; their benefits and disadvantages will be further discussed in upcoming sections.

5.2 Architectural Component Overview

Based on the general RASCAL framework presented in Chapter Three along with the above organizational overview, this section presents details about specific aspects of the MSpace prototype. First, a global perspective is offered to show how the general architecture maps to the prototype implementation, followed by discussion of individual components.

5.2.1 MSpace: The Global Perspective

The RASCAL framework presented in Chapter Three offered both a generic architecture and a methodology for the organization of collaborative media space systems. Figure 5.3 presents how that architecture can be mapped to a generic implementation in which "slots" for different technologies to be used within an actual

Architectural Component		Technology	
Client Point of Presence		OS Supported Infrastructure	
CCS	Front End — CCSfe	OS-Supported Infrastructure	Active Object Executable Models
	CMSS Behavioural Backbone	Active Object/Executable Models	

Table 5.1: Architecture Technology Allocation

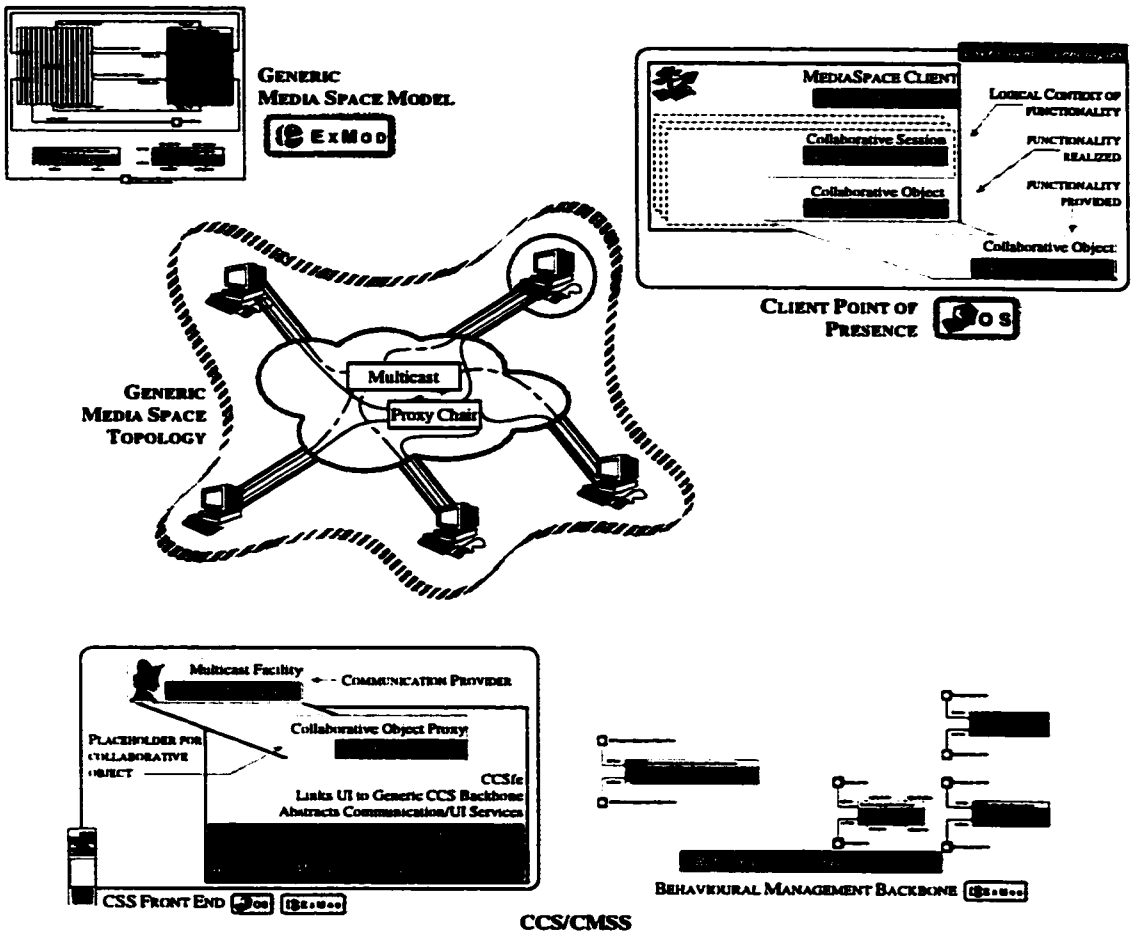


Figure 5.3: Global Perspective of Model to Implementation Mapping

system. The generic media space topology in the centre of the picture is that first shown in Chapter Three, and radiating from it are the three primary system-level architectural entities that can be addressed via disparate technologies:

1. The Generic Media Space Model
2. CCS/CMSS
3. Client Points of Presence

The Generic Media Space Model. This top-level model is used to outline the media space's architectural components (both hardware and software) and their interconnectivity relative to the RASCAL framework. As such, it functions as a "container" which represents the whole media space and references to entities within it depict the generic organization of the space independent of a specific usage or configuration. In terms of the prototype, the generic media space model is represented as an executable model using the ROOM methodology via the ObjecTime Developer CASE tool. The visual design metaphor offered by ObjecTime (whose basics are outlined in Table 5.2) uses an object-oriented approach (introduced earlier in Section 2.2.4 and further outlined

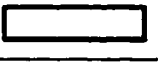
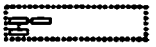
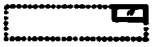
Domain	Symbol	Specific Meaning	Comment	
Active Object Execution via Actor		Fixed (Static)	Created statically as a fixed part of architecture	
		Optional (Dynamic)	Created dynamically as an optional part of the architecture	
		Imported	Slot for insertion of an actor into a specific role	Said actor is already existent
		Optional Imported		Said actor is dynamically created
		Structural Nesting	Given actor (any type) contains other actors within it	
Multiplicity via Replication Factor		Group of Actors	Number of a given actor type; maximum number if optional actor	
		Multiple Ports	Number of a given port	
Communication by Connected Ports		Connection Between Individual Ports	Connection between specific instances of a port on specific actors	
		Connection Between Multiple Ports	Connection between groups of actors, one connection per	

Table 5.2: ROOM Structural Diagram Basics

in Table 5.3) as the means by which the system designer/architect constructs his/her model(s). The top-level definition of the MSpace prototype is based on this approach along with a set of classes that is one possible realization of the generic RASCAL framework

Such an approach is shown in Figure 5.4, which illustrates the top-level structural definition of the MSpace prototype. In this model, the media space is created as a single instance of the actor class *RASCALMediaSpace* and functions as a container (via structural containment) for all other components in the system. These include physical entities such as the Coordinated Communications Subsystem (*mspaceCCS*) and a set of "point of presence" client devices/workstations (*mspaceUserClientStations*) as well as more "logical" entities, including collaborative objects and behavioural patterns, represented by *mspaceObjects* and *mspacePatterns* respectively.

Class	Purpose	Nestable	Inheritance	Comment/Explanation
Actor	Active Object Specification and Incarnation	Yes	Yes	Defines entities that have their own computational capabilities/run-time behaviour. Allows multiple incarnations of the same type of entity based on well-known structural and behavioural definitions
Protocol	Port/Connection Specification	No	Yes	Defines the connectivity between actors within the system. Actors can only communicate via specified ports and their connections to other actors; the definition of a particular connection's characteristics is defined by the protocol class.
Data	Data	Yes	Yes	Forms the traditional notion of data and passive objects within the system; used by the actors within computations (e.g. variables, constants)

Table 5.3: ObjecTime Classes

Further explanation of Figure 5.4's annotations can be found in Table 5.4 while the class/entity name prefixes used in this diagram (and throughout the rest of the chapter) are explained in Table 5.5.

Through the use of structural containment, complex entities are defined as composites of role/task-specific active objects. For example, the CCS (as shown in Figure 5.5) contains a multicast facility and the Proxy Chair, an arrangement which maps directly to the generic architectural topology outlined in Chapter Three. In addition to these physical elements, logical entities such as collaborative behavioural patterns and objects (i.e. *mspacePatterns* and *mspaceObjects*) are also built in a similar manner. Unlike the physical components, these logical entities are defined "globally" as a system pool of entities that are used "locally" within various role-specific contexts. Consequently, they have no specific connectivity to other elements at the definition level, but do within the role-specific slot they occupy during execution. Additionally, this arrangement corresponds to the use of the behavioural database, presented as part of the RASCAL topology.

Chair form the basis for the CMSS, which provides the communication subsystem's actual functionality. For purposes of the prototype, the Proxy Chair is represented as a single static entity to match its logical definition. While a simple implementation is sufficient for illustrative purposes, more complex realizations of the Proxy can be retrofit into the current model through evolutionary changes to the Proxy implementation. Doing so is supported through the use of specific actor classes to create and isolate such entities from their those that utilize them. That is, the CCS is based on the class RBE_CCS while the multicast facility is based on

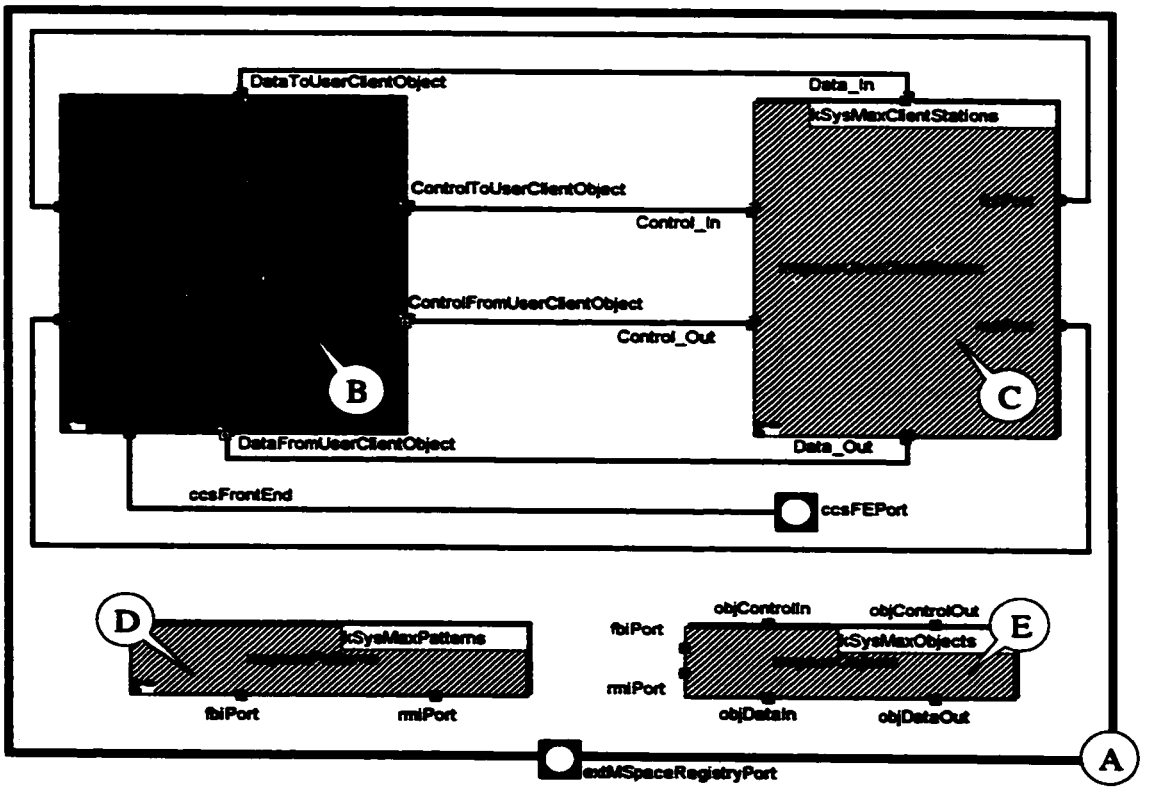


Figure 5.4: Annotated Structural Model of the RASCAL MSpace Prototype

Figure 5.4	Label	Entity Name	Construction	Ordinality	Comment
	A	RascalMediaSpace	Static	Single	The entire media space system
	B	mspaceCCS	Static	Single	The combined CCS/CMSS
	C	mpaceUserClientStations	Dynamic	Multiple	Users' physical locations
	D	mpacePatterns	Dynamic	Multiple	System-wide pool of collaborative patterns
	E	mspaceObjects	Dynamic	Multiple	System-wide pool of collaborative objects

Table 5.4: Legend for Annotated MSpace Structural Model

RBE_MulticastFacility and the Proxy Chair upon RBE_ProxyChair. The RBE_ prefix (again see Table 5.5) refers to the basic and requisite nature of the said entity to the framework. More specifics on the CCS/CMSS as the means for dealing with behavioural management will be offered in upcoming sections.

Client Points of Presence. The set of entities known as the "points of presence" represent client devices (i.e. workstations or similar end-user device) that collaborators physically utilize to interact with other collaborators within the media space. Represented as a collection of *mspaceUserClientStations* of class *RBE_UserClient*, these are the access points where users utilize collaborative objects within the bounds of a logical collaborative session. As such, they act as composite sources/sinks for events that are actually generated by specific, fine-grain entities (i.e. the collaborative objects). In a simulation-oriented environment, these entities could be used to isolate and provide traffic generation facilities of the appropriate type. In a system that provides actual collaborative functionality, however, these entities serve as placeholders and interfaces to the specific application functionality provided via disparate technologies (such as the OLE-based tools in MSpace). More specifics on example MSpace client points of presence will be offered in upcoming sections.

5.2.2 MSpace Behavioural Management: The CCS, CMSS & the CCSfe

Behavioural management within MSpace is addressed using a complex mix of both modelling and

Prefix	Applied To	Interpretation/Importance
mspace	Actors and/or Ports	Top-Level Prototype Structural Definition
rms		Rascal Media Space
ccs		Coordinated Communications Subsystem
rs		Reflective Service
fbi	Ports	Functional Base Interface
rmi		Reflective Meta Interface
RASCAL	Classes	Framework-Level Definition
RBE_		Rascal Base Entity
RBS_		Rascal Base Service

Table 5.5: MSpace Naming Convention Prefixes

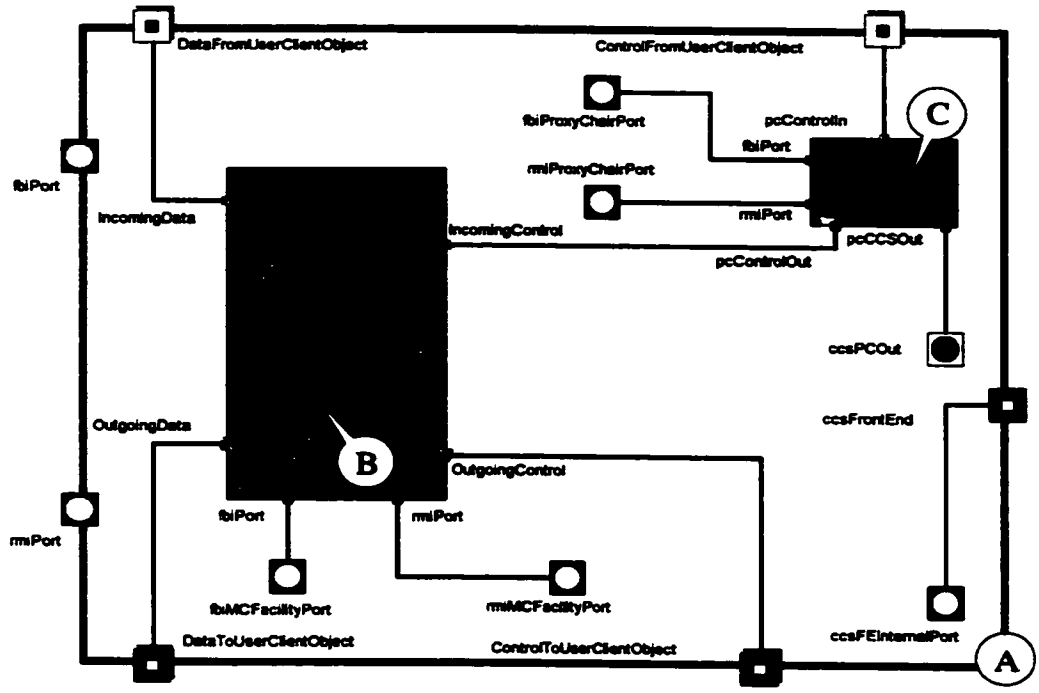


Figure 5.5: Annotated Structural Model of the MSpace CCS

Figure 5.5	Label	Entity Name	Construction	Ordinality	Comment
	A	mspaceCCS	Static	Single	The combined CCS/CMSS
	B	ccsMulticastFacility	Static	Single	Models/provides for multicast communication
	C	ccsProxyChair	Static	Single	Models the Proxy Chair – i.e. the media space's overall coordinator

Table 5.6: Legend for Annotated MSpace CCS Structural Model

implementation techniques. In accordance with the basic RASCAL framework, behavioural management is primarily addressed by the CCS/CMSS behavioural backbone. This backbone, which is built using the active object/executable model paradigm vis-à-vis ObjecTime, however, must interface with those technologies used within particular client point of presence implementations. In order to transition between these two disparate approaches, MSpace utilizes a transitional point called the *CCSfe*, i.e. the "CCS front end". The CCSfe provides the translation between the backbone technologies and those particular to the client point of presence implementations used within the prototype (that is, the MSpace client application and the collaborative tools used within it).

The following section now details the implementation of these various MSpace components. Following that, the particulars of the client point of presence application and its relationship to these CCSfe is illustrated.

5.2.2.1 CCS/CMSS: Active Behavioural Modelling and Simulation

The MSpace behavioural backbone is realized as an ObjecTime-based ROOM executable model whose components are active objects based on the generic RASCAL class framework. Thus, the entities modelled as part of the backbone can take advantage of the object-oriented, actor-based implementation and simulation[10] facilities of the ObjecTime toolset.

Consequently , by virtue of their own thread of execution, both "physical" and "logical" entities can perform computation rather than being strictly passive objects (Table 5.7). Those such as the Proxy Chair and the Multicast Facility utilize their active nature to perform their functionality, such as to coordinate/transmit messages within the CCS. The behaviour of these structural entities is therefore embodied by their execution relative to their functional role within the system. Their modelling as an active object, therefore, is straightforward and easily understood.

Conversely, logical constructs such as behaviours and collaborative sessions are not usually tangible entities. Rather, they are taken as inherent aspects of processes involving the use of "real things", such as specific applications or communications network. As a realization of the RASCAL framework, however, MSpace uses first-class representations of these logical entities. That is, each behavioural pattern is defined and incarnated as its own independent executable unit (vis-à-vis the active object/actor paradigm). As part of this execution, each pattern can perform its own computation, which can involve monitoring its own behavioural performance, making decisions based on that analysis, and subsequently affecting changes in response to that evaluation.

This ability to monitor, analyze and adjust is facilitated through the use reflection and a meta-level architecture. By forming the basis of a dynamic organizational methodology, the performance and role of entities can be

<table>
<tr><th colspan="3">Entity</th><th colspan="3">Specialization</th><th rowspan="2">Provision via Thread of Execution</th></tr>
<tr><th colspan="2">Type</th><th>Example</th><th>Subclassing</th><th>Design Time</th><th>Run Time</th></tr>
<tr><td rowspan="2">Physical</td><td>Structural</td><td>Application Tools, Media</td><td>Implementation Technology-Dependent</td><td>Inherent in Implementation Code / Logic</td><td>Implementation Technology-Dependent</td><td>Independent Functional Execution</td></tr>
<tr><td></td><td>Topological Constructs (e.g. CCS)</td><td>via ROOM Classes Model</td><td>Structural Containment</td><td>Slots & Actor Importation</td><td>Independent Construct Behaviour and Management</td></tr>
<tr><td rowspan="2">Logical</td><td rowspan="2">Behavioural</td><td>Collaborative Scenario Definition</td><td rowspan="2">via ROOM Class Model</td><td rowspan="2">via Hierarchical FSM</td><td rowspan="2">Slots & Actor Importation</td><td rowspan="2">First Class, Explicit and Independent Realization of a Behaviour as Executable Logic</td></tr>
<tr><td>Synchronization Algorithm Specification</td></tr>
</table>

Table 5.7: Entity Specialization and Execution

[10] Discussion of the availability and utility of the MSpace/ObjecTime simulation facilities will be deferred until Chapter Six, due to its "evaluatory" nature.

adapted relative to the larger computational system at different levels of abstraction. Therefore, by realizing logical entities, such as behaviours, policies and collaborative sessions, as concrete computationally active objects, they can monitor, analyze and adjust the performance and role of entities relative to the collaborative effort and the ability of the system to meet its intended goal.

5.2.2.1.1 Behavioural Building Blocks: Components and Patterns

As shown earlier, a key aspect of the RASCAL framework is its parallel approach to collaboration. That is, the traditional functional methodology is augmented with one in which behavioural components form the fundamental "building blocks" used to describe collaborative activities. Within MSpace, each of these individual components are self-contained behavioural specifications implemented as an ObjecTime actor. Accordingly, the execution logic of each actor, and therefore the executable behavioural rules that it uses to regulate collaborative events, is provided by its own finite state machine (i.e. FSM). At the lowest level of system design, a behaviour describes the basic functionality of a specific application or functional component via the rudimentary entry points for tasks that entity can perform. That is, the FSM can be seen as an event-to-function mapping mechanism. Subsequent to that, behavioural components are hierarchically organized such that the domain of a higher meta-level (more abstract) behaviour (i.e. what it computes about) is defined as a lower-level (less abstract) behaviour. Consequently, behaviours can monitor, analyze and adjust the performance of other behaviour(s), forming the foundation for a self-adaptive behavioural management system.

In addition to hierarchical linkages, RASCAL also utilizes behavioural groupings known as collaborative associations (see Chapter Three). These CAs exist as patterns of individual behaviour (component) specifications which, when assembled into semantically and syntactically connected groups that work together, meet particular needs within the collaboration. However, rather than place the complexity of numerous and/or diverse behaviour specifications within a single actor, a behavioural pattern is represented as a group of interacting actors; therefore, effectively forming a behavioural subsystem. Each behavioural pattern is constituted as a parent behavioural component (and therefore actor) which administers and communicates with its base-level behaviours (actors) to ensure the intended correlation and consistency between levels of the specific pattern. The composition of actors as managed and interacting hierarchies (which communicate with each other via message passing) form the of basis for active behavioural patterns which define higher-level behaviours which can be used to specify increasingly complex collaborative interactions. That is, by recursively taking behavioural patterns as components within other behavioural specifications, complex behaviours can be constructed using components and patterns in an interchangeable manner, as nodes in the collaboration's corresponding collaborative behavioural tree (as presented in earlier chapters). Through relating these entities via reflection, a standard approach to organization and interaction can be provided; and consequently, it is through this standardized approach that the ability to customize not only becomes legitimate, but an integral part of managing both the structure and the behaviour of the system.

5.2.2.1.2 Customizing Behaviours: Both Inside and Outside the Blocks

The initial design of any system is typically based on the requirements of its intended users. Therefore, the designer effectively "customizes" the system using suitable components, functionalities and behavioural specifications to meet user needs at the time the system is built. However, as times goes on, users typically require different functionality and different behaviour from the system. That is, in collaborative terms, just as the collaborative process itself changes over time, so too must a collaborative system. Consequently, the ability to change both functional and behavioural aspects is a necessary part of system evolution. As an implementation of the RASCAL framework, MSpace uses (active) object-orientation combined with reflection and its intrinsic support for tailorability to offer the necessary flexibility for both functional and behavioural evolution. In this section, emphasis is concentrated on behavioural issues.

As outlined in Table 5.7, MSpace uses various specialization techniques to take advantage of pre-existing behavioural specifications and build upon them. Design-time specialization is provided through the ability to specify custom finite state machine logic on a per actor basis combined with the ability to specialize an actor via use of ObjecTime's actor classes. Run-time behavioural specialization, which allows changes to the system during execution, is provided through "behavioural substitution" via dynamic actor importation (as provided by ObjecTime). This flexibility is achieved by substituting and composing behaviours using *behavioural slots* in which separately defined behaviours (via their actors) can be inserted. Pools of collaborative behaviours are defined globally at the system level (see Figure 5.4) and are dynamically imported into role-specific slots which define their execution context. By doing so, the same behavioural entity can serve as the definition as a top-level session (scenario) specification (such as in Figure 5.6) or as a portion of a more complex behaviour (as

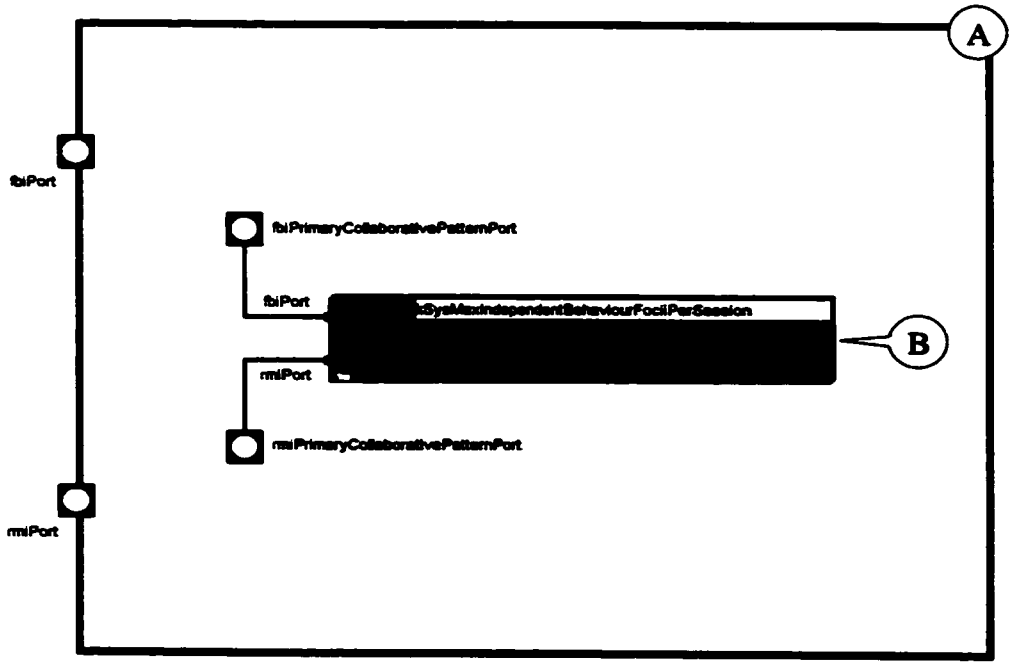


Figure 5.6: Annotated Structural View of Collaborative Session

Figure 5.6	Label	Entity Name	Construction	Ordinality	Comment
	A	rmsCollaborativeSession	—	—	Each collaborative session exists as a collection of individual top-level collaborative patterns
	B	rmsPrimaryCollaborative Patterns	Dynamic/Slot	Multiple	Specific collaborative patterns are imported as part of the session definition as relevant to execution of the collaboration

Table 5.8: Legend for Annotated Collaborative Session Structural Model

outlined in Figure 5.7). These figures are annotated in Table 5.8 and Table 5.9 respectively; further details of these particular constructs will be provided in upcoming sections.

As an example, consider a run-time behavioural modification typical of that outlined in the business meeting scenario in Chapter Four. Shown in Figure 5.8, the left frame shows the notion of behaviour replacement using conceptual clouds of lecture and brainstorming behaviour specifications while the right frame shows the corresponding "actor and role" approach used within MSpace, vis-à-vis a behaviour "slot". By allowing different actors, each with their own particular execution logic, to execute in the context of a scenario-specific role, the flow of event generated by user activity would be regulated according to the rules of the actor currently executing in the slot. Consequently, by changing the actor, the behavioural rules which govern that particular part of the scenario's behavioural definition would change, therefore resulting in a change in the behaviour expected and accepted by the system. However, two important issues remain: (1) what constitutes a behaviour; and (2) how does reflection fit.

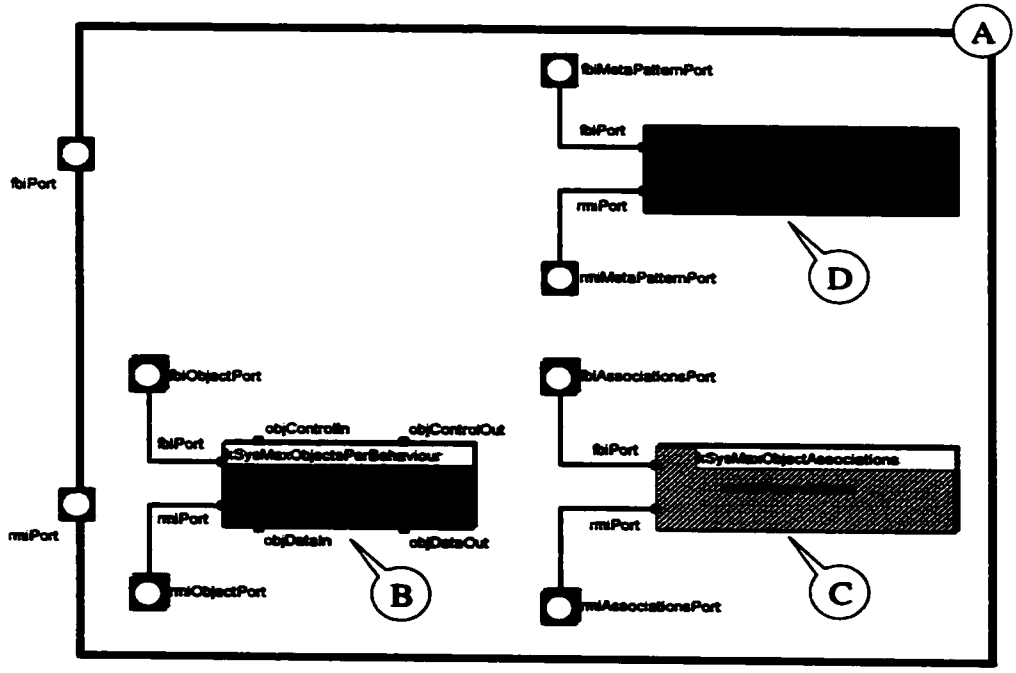


Figure 5.7: Annotated Structural View of Collaborative (Behavioural) Pattern

Figure 5.7	Label	Entity Name	Construction	Ordinality	Comment
	A	rmsCollaborativePattern	—	—	Each collaborative pattern exists as an organized grouping of specific the following entities
	B	mspaceObjects	Dynamic/Slot	Multiple	Objects governed by this pattern
	C	mspaceAssocations	Dynamic	Multiple	Relation logic between objects
	D	rmsMetaPattern	Slot	Single	Optional Meta-level Definition

Table 5.9: Legend for Annotated Collaborative (Behavioural) Pattern Structural Model

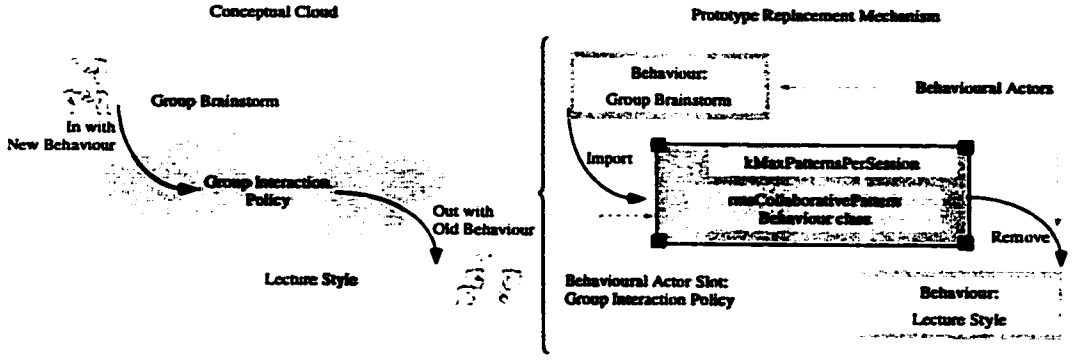


Figure 5.8: Realizing Behavioural Changes via Behavioural Slots

5.2.2.1.3 Inside the Blocks: Constituting Sessions and Patterns

Based on the RASCAL behaviour construct outlined in Chapter Three, MSpace defines two types of entities to regulate collaborative scope: (1) Sessions; and (2) Patterns. Outlined in Figure 5.6 and Table 5.8, a session is a top-level behaviour which specifies the goal of the collaborative scenario. A session can contain multiple top-level patterns in order to support a complex scenario conceptually having multiple independent tasks that are in a peer-to-peer relationship. That is, a scenario can be defined as a group of independent (or "equally important") behaviours that are addressed separately in order to meet the scenario's overall goals. Typically, however, a single behaviour would be used to define the overall behaviour (and goal) of the collaboration. This typical situation can therefore be regarded as a straight-forward subset of the more complex generic approach outlined previously. In fact, a collaborative session is effectively a "container" for the specific behavioural patterns (as discussed below); it therefore has no specific function other than to allow for independence between top-level behaviours.

In contrast, patterns are the primary behavioural representation used in MSpace. As presented in Figure 5.7, a pattern is itself a compound entity. While the declaration, incarnation and context-specific usage of a pattern (as behavioural components) were discussed in the previous section, its internal elements (as outlined in Table 5.9) allow a pattern to be an extremely flexible and general construct. The objects referenced within a pattern are "slots"; this allow specific collaborative objects from the "object pool" defined at the system level (*mspaceObjects* in Figure 5.4) to be dynamically inserted into the pattern which is to manage their behaviour.

Like the behavioural pool, this approach supports the notion of system-wide collaborative object database for managing collaborative functionality as a separate yet integrated part of the whole collaborative process.

Within each pattern, a group of optional *Associations* can be defined strictly within the bounds of the pattern to defined the relationship logic of the pattern. As a dynamically instantiated actor, each association is defined using the actor class framework and can be added/removed as required over evolution of the pattern; it also limits implementation details to a particular FSM. This enables classes of associations to be defined in which refinements to a basic relationship can be implemented simply as a subclass of the association's actor class. It is the responsibility of the behaviour actor to define the logic and interaction between the objects and the associations within the behaviour. Doing so is not always strictly static; nor does it always involve the same set of actions across different associations; therefore, provisions for addressing this issue must be provided. These will be illustrated in subsequent sections.

Finally, the third part of a pattern is its "meta pattern", which defines its (optional) meta level. By providing an optional meta level, each pattern can potentially be used at any point within a reflective tower (which define RASCAL's hierarchical behaviour specifications). Again, it is the responsibility of the given behaviour to provide the appropriate interaction with the meta level via use of reflection (discussed next).

5.2.2.1.4 Applying the Reflective Approach

As an example of the RASCAL framework, the organization of MSpace is designed to take advantage of a reflective meta-level architecture. In particular, the ability of the behavioural components to participate in reflective computations is integral to the system's ability to monitor, analyze and modify itself during the collaborative process.

To describe how reflection is applied within MSpace, three basic areas are briefly overviewed: (1) the basis of the meta-level architecture; (2) the essence of the meta object protocol; and (3) the provisioning of reflection within the environment.

5.2.2.1.4.1 The Basis for the Meta-Level Architecture

All entities within the MSpace model are based on the object-oriented paradigm, using the class organization framework provided by ObjecTime. Additionally, because the essence of RASCAL is to provide a reflective meta-level architecture, all entities require the means for being incorporated within a meta-level architecture as well as being able to perform reflective acts. Therefore, these capabilities are provided using the class derivation facilities of ObjecTime in a manner analogous to the generic RASCAL class framework outlined in Chapter Three.

In MSpace, all entities are derived from the root class *RASCALBase*. In this root class, the ability to perform reflective acts is defined, and in the case of MSpace involves entity interaction with the provider of the reflection service (since the development tools utilized did not inherently support reflection). Therefore, *all* derived objects are reflective in that they inherit access to the reflective mechanism provided by the

RASCALReflectionService (as will be presented below). Access to this service is provided using an explicit meta interface (as per [KP96]) called the *Reflective Meta Interface*, or *RMI*. The principle is to provide location and entity-independent support for reflection. This interface parallels the standard functional interface (which handles "standard" application/object-specific functionality that is the essence of that object's base level). For this reason, this interface is called the *FBI*, or *Functional Base Interface*. Consequently, it is through use of the ObjecTime actor paradigm combined with the above support for reflection that the REXS stereotype (as presented in UML discussion of the RASCAL framework in Chapter Three) is provided.

5.2.2.1.4.2 The Role and Realization of the Meta Object Protocol

As presented in Chapter Three, the meta object protocol defines how an entity interacts with its meta level(s); or more precisely, a MOP governs an entity's interactions with its meta object(s). Therefore, in terms of applying reflection to behavioural definition and management within MSpace, the role of the MOP is the facilitate the interaction and control of various behavioural components and patterns.

As part of MSpace's behavioural backbone, MOPs are defined using ObjecTime's protocol class facilities. As these protocol classes define the types of messages actors can send between each other, the meta object protocol is the basis for the exchange of information on the Reflective Meta Interface (RMI) ports which are part of every MSpace entity. That is, RMI provides a standard routing mechanism for the development of a (family of) meta object protocols through the use of protocol subclassing. Consequently, the system designer can opt to provide different MOP granularities, such as a single MOP that deals with multiple meta-level issues or a series of MOPS, each of which is specialized with respect to a specific metaobject's concerns. For example, as illustrated by the various meta-level issues that constitute the various meta spaces outlined in Chapter Three, a single "CARE MOP" which combines all aspects of CA meta object interaction. Alternatively, an organized collection of aspect-specific MOPs could be defined, possibly based on a common CARE protocol parent.

5.2.2.1.4.3 The Provisioning of Reflection

The development of architectures and systems based on the reflective paradigm typically presuppose one thing: that the ability and infrastructure necessary to perform reflective acts are intrinsically a part of the underlying computational system. More often than not, unfortunately, this is not the case. And as in the case of MSpace, the underlying technologies used were not natively reflective. Consequently, reflective capabilities had to be explicitly provided.

The approach taken in MSpace[11] is again based on the active object paradigm, through the use of an ObjecTime *service* [Obj97]. Such a service differs from the actor-specific approach in two main aspects:

---

[11] That is, as related to the behavioural backbone (CCS/CMSS) in particular. Reflection as applied to user-level collaborative tools and applications will be discussed in subsequent sections.

- It is not represented in a graphical manner like the other structural entities.
- A service allows multiple actors to communicate with and utilize its functionality without the need to explicitly embed and connect specific instances via ports.

As an example, ObjecTime provides access to time-related functionality through a built-in "timer service". Using what is known as *Service Access Point*s and *Service Provision Point*s (SAPs and SPPs respectively) [Obj97], actors can access timing functions without the need to graphically position a "timer" in each part of the system that needs its facilities. Because timing issues are considered a basic requirement within many systems, timing facilities are provided in a manner that eases their use, management and implementation. Therefore, in the same vein, utilizing a service rather than individual actors to provide reflective capabilities is justifiable as it:

- Offers a more intuitive functional approach, acting more as a general facility which can be used rather than as a specific entity with which to interact.
- Simplifies entity connectivity, since there is no need to connect specific ports.
- Enables a more transparent approach to providing a system-level facility (as though the underlying system was providing the service) rather than specifically requiring each entity to explicitly deal with its own local provider.
- Provides a standard location to define, implement, measure and tune facilities independent of system topology.

Called the *RASCALReflectionService*, the MSpace reflection service offers backbone entities the ability to request a reflective computation/act be performed on their behalf. Because the essence of performing reflection is the ability to examine, modify and intercede in the organization and execution of a system, the MSpace reflective service isolates the modification of actor/slot pairings within ObjecTime's structural models. Therefore, any specific technological issues that are required to actually enable the reflection to occur (such as switching the slotted actors) can be isolated. It also facilitates the potential to provide evolutionary change in the reflective abilities due to any potential changes in backbone technology.

Outlined in Table 5.10 (with icons shown in Table 5.11), the *RASCALReflectionService* is shown as a three level hierarchical state machine that responds to *Reify* and *Deify* events. The top level of the FSM consists of three states that allow for the system to place any startup and teardown logic for the reflective infrastructure independent of the main operational state (*RSActive*). Once active, the second level of the FSM details whether an object is computing at its base level or performing meta-level computations. Subsequent reflective acts, which would constitute the formulation of a reflective tower are shown in the third level of the FSM. It is the responsibility of the logic within the third level FSM to track the level of the reflective tower currently being computed. This provides an intuitive and finite representation of a potentially infinite reflective structure.

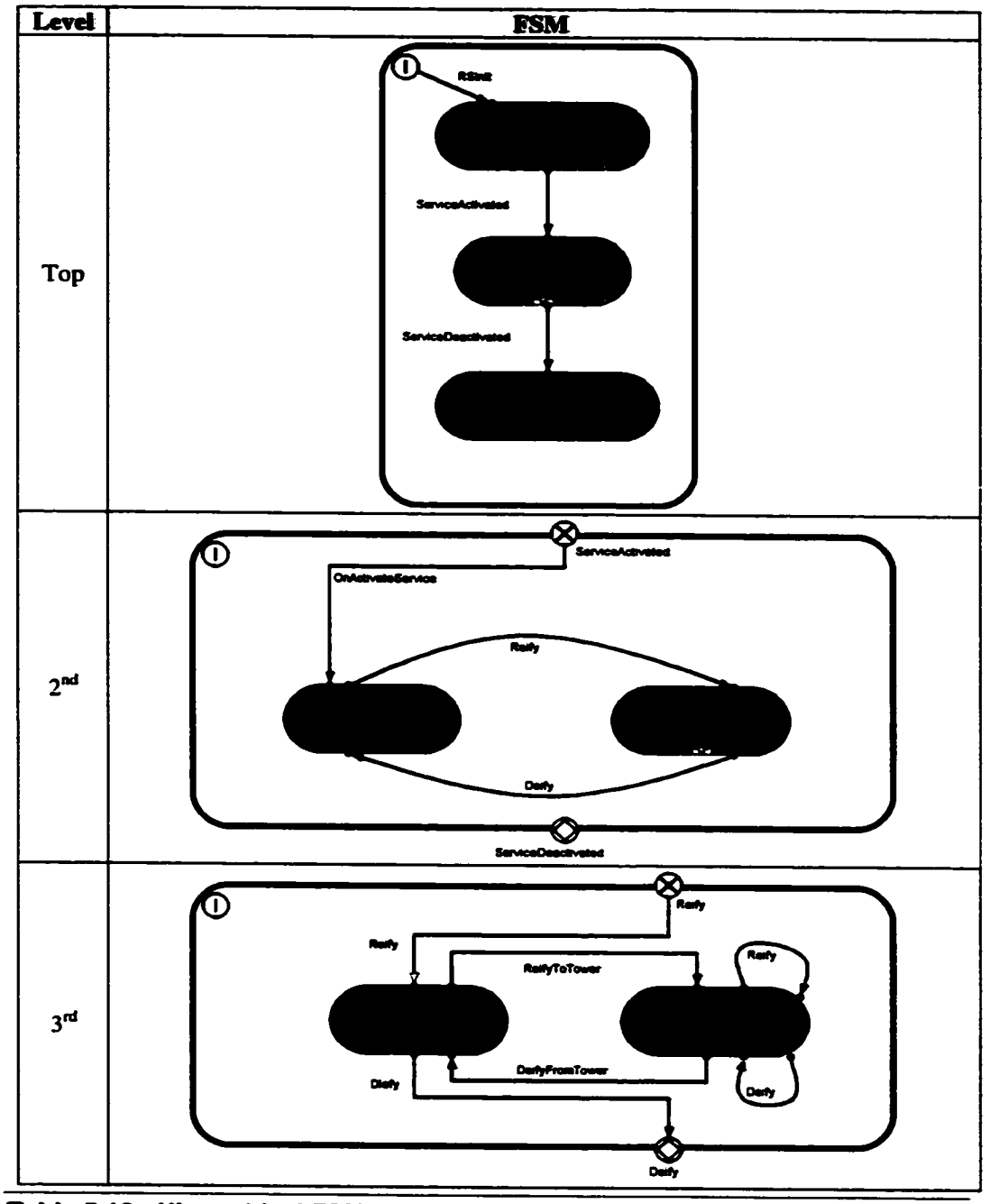


Table 5.10: Hierarchical FSM for MSpace Reflection Service

As an example, consider the formulation of a given object's reflective behavioural tower, outlined in Figure 5.9. As an object is executing in its base state, its behavioural status must be queried in order to ensure its correct operation. Hence, a Reify event triggers a switch in the object's processing from its functional base level to its behavioural meta level. The logic at the behavioural level specifies the behavioural options and sequences of functional operations that entity can perform by itself. However, the system is aware that this object is part of a collaborative grouping that specifies a joint behavioural specification. Therefore, a synchronization behavioural meta-object (for its individual synchronization policy) and a coordination behavioural meta-object (for group

Domain	Symbol	Specific Meaning	Comment
State		Individual leaf state	Standard notion of state (non-composite)
		Hierarchical parent with sub-states	Composite state which allows nesting of sub-states within it
Transition		Transition between states	Move from one state to another based on a triggering event; perform code body associated with said transition
		Initial Transition	Fired upon first entry into a state
		Hierarchical Transition Entry	Entry point for sub-state
		Hierarchical Transition Exit	Exit point for sub-state
Choice Point		Boolean Selection Transition Point	Allows a single transition to conditionally branch

Table 5.11: ROOM/ObjecTime Behavioural Diagram (FSM) Basics

coordination) are two additional levels in the behavioural hierarchy. Based on reification to the synchronization level, that policy determines that the object is functioning as intended; however, when it reifies itself to the group coordination policy meta level, that level determines that the group is not coordinating itself as intended. As a remedy, it determines that the synchronization behavioural specification is not appropriate and therefore switches the synchronization policy to one more appropriate in the attempt to satisfy both the lower-level synchronization requirements as well as the group coordination effort. Therefore, via reflection the meta level decides (based on its internal logic) whether lower-level behavioural modifications are necessary. If so, it would remove the said behave from the behavioural slot and import the appropriate actor to replace it. Then upon deification (i.e. a Deify event), the new behaviour would take effect (returning to the thread of execution belonging to the actor in the slot).

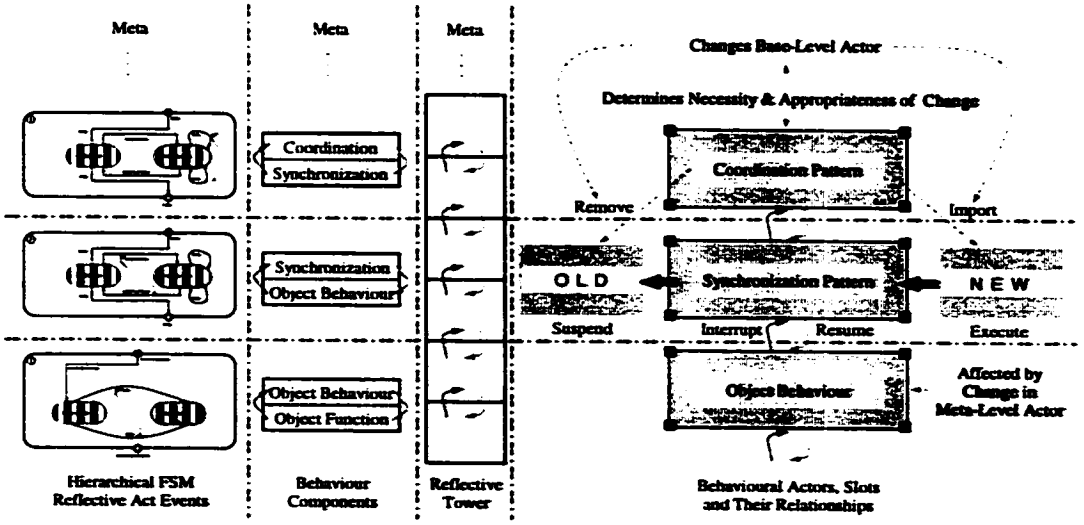


Figure 5.9: Representing A Behavioural Tower – FSM, Components, Actors

5.2.2.2 CCSfe: The Technological Transition Point

The CCSfe is a general purpose front-end to the CCS which can be used to map differences between the technological approaches used to implement the behavioural backbone and those used to implement the user points of presence and associated collaborative tools. The CCSfe is not a specific part of the RASCAL framework per se, as its necessity is relative to the technologies used to implement a system rather than the framework itself.

Therefore, in the case of MSpace, the CCSfe is used to map differences between and connect the OS-specific user interface technologies and the ObjecTime-based behavioural backbone. Hence, it is where the idiosyncrasies of the various implementation technologies and their relative suitability/compatibility come to the fore. It can therefore be seen as a means of isolating technological inter-dependency, functioning as an interface between them. Consequently, the MSpace CCSfe utilizes a proxy-like approach (refer back to Figure 5.2) in providing it provides a convenient place to monitor the interactions between the two parts of the system. This approach is shown in Figure 5.10 in which the same technologies as used in the MSpace client (discussed shortly) are used to connect the ObjecTime portion of the system to the distributed user clients. As an example of possible management interfaces, the MSpace CCSfe offers a "report-oriented" visual tracking of events as they occur relative to the creation of a collaborative session and collaborative objects within it.

5.2.3 MSpace Client: The User Experience

As part of the conceptual prototype, a point of presence client application was developed to illustrate one possible approach to user interface design. The client's design was based on the available technologies and with the desire to mirror the integrated nature of the media space concept. Detailed previously in Chapter Two, the MSpace client uses technologies based around the use of OLE technology. Mirroring the flexibility of the RASCAL framework itself, the prototype's user interface is illustrated by an assortment of screen shots shown in Table 5.12.

In each instance, the prototype's main (application) window represents the participants' view of the whole media space, functioning as a container for different collaborative sessions (realized as child windows within

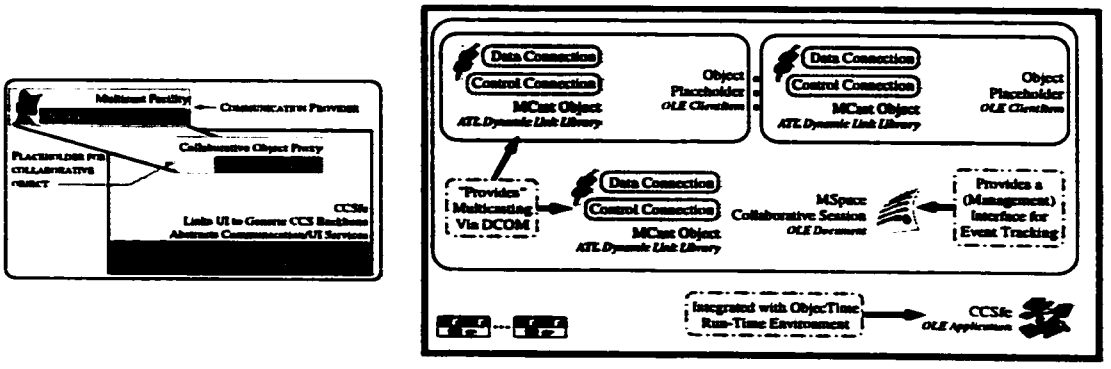


Figure 5.10: MSpace CCSfe Technology Organization and Rationale

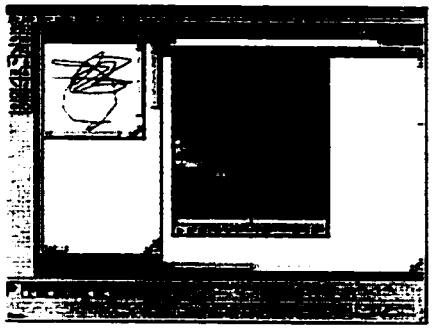
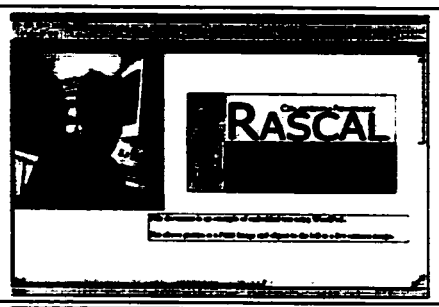
Domain	Sample User Interface	Comment/Explanation
Multiple Individual Media Sessions		Three individual collaborative sessions: 1. Movie player 2. Drawing application 3. Unknown (obscured) The drawing application is currently receiving input and its interface is active. The other sessions are still active but their interface is not receiving input.
Mixed-Mode Session Utilizing Real-Time Video Capture		A single collaborative session multiple media: 1. Live video capture via a camera 2. Graphic bitmap 3. Text document
Mixed-Mode Session Utilizing Synthetic 3D Animation		A single collaborative session multiple media: 1. Text document 2. Graphic bitmap 3. A 3D navigatable virtual world

Table 5.12: User Interface Screen Shots of Example Scenarios

it). Each session consists of objects that represent media and applications utilized within the collaboration. By utilizing OLE technology, a wide variety of objects can be inserted into the space using a component-based approach akin to implementational reflection. Users familiar with the "Insert Object" facility found in applications such as Microsoft Word™ will be well-acquainted with this technique.

Specific functionality is dynamically deployed using the insertion of an OLE Client Item (representing the user interface end of an OLE client/server pair) into a containing OLE Document (representing the collaborative session). The functionality of the actual component is localized within a separate OLE server while its user interface is realized in the bounds of the document using in-place activation. The organization of these OLE constructs in relation to the generic client organization is outlined in Figure 5.11.

Therefore, each object shown in the previous screen shots is transparently enabled by its own specific application (as offered via an OLE server) and provides its functionality in a seamless and integrated fashion within a unified user interface. Therefore, rather than using disparate applications with noticeably disjoint user interfaces, the result is an "applet-like" approach within a single collaborative environment.

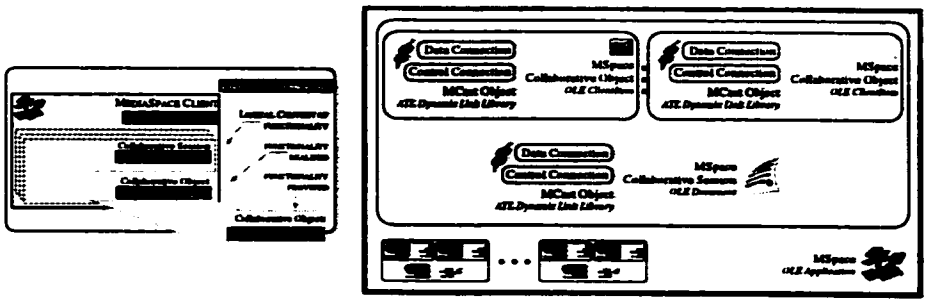


Figure 5.11: MSpace Client Technology Organization

As the various applications and tools are utilized within the user interface, the corresponding events generated by their manipulation is intercepted as passed to the behavioural backbone to be managed relative to the intent of the collaborative scenario. Such application tools can include existing OLE-capable applications (i.e. OLE servers) as well as those custom written for such purposes[12]. Examples of custom servers attempted as part of this research include real-time audio/video microphone/camera capture, a modified version of the Windows Media Player and a remote telepointer application. Each of these tools were written in C++ and utilized a non-trivial combination of technologies including MFC, ATL, ActiveX, D/COM and DirectX.

Unfortunately, an additional complexity resulted from the need to enable multicasting within the system. Due to OLE's client/server approach, a specific entity was developed to facilitate sending and receiving multicast data (and control messages) on a per entity basis. This *MCast* object, shown in Figure 5.11, is an embedded ActiveX control (based on the custom *MCaster* dynamic link library written in C++ using ATL) that provides for multicast connections via DCOM *connection points* [Gri97] [Tem97]. This entity is embedded in any object that needed to send or receive multicast data, including collaborative objects and sessions at the client as well as their placeholders within the CCSfe.

5.2.4 MSpace and Beyond

Based on the overview of technologies used within different MSpace components, Figure 5.12 shows the correspondence between various implementation technologies and their representation within different parts of the implementation. As part of its development, a class hierarchy based on the generic RASCAL framework was created to guide construction of the actual MSpace prototype. While portions of this hierarchy were discussed earlier in this chapter, more details on specific classes, their organization and utilization as a basis for future systems are left for discussion in Chapter Six.

---

[12] Several key issues relevant to the suitability of OLE for use within the context of a real-time collaborative system became obvious during prototype development. Their discussion is deferred to Chapter Six due to their evaluatory nature.

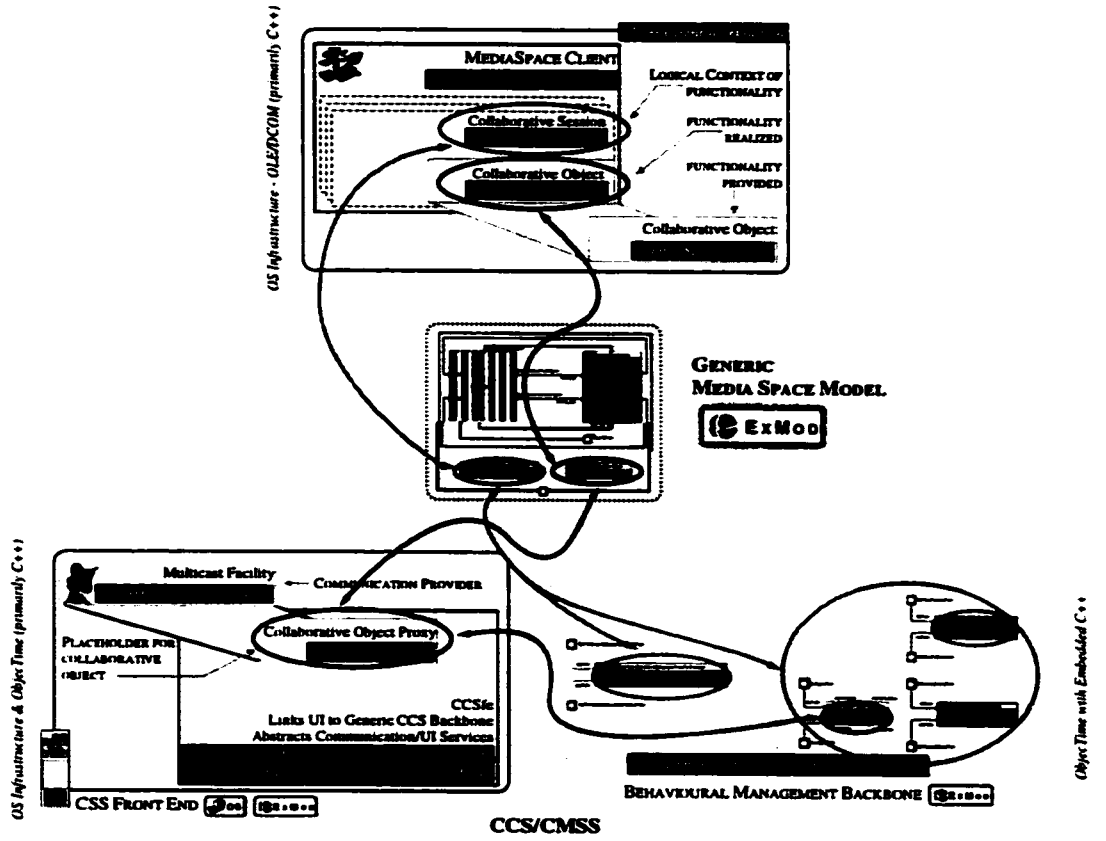


Figure 5.12: Correspondence Between MSpace Entities Relative to RASCAL

5.3 Summary

This chapter has presented an overview of the MSpace conceptual prototype which illustrates one possible approach to implementation of the RASCAL framework. Specific MSpace implementation details were highlighted as well as how they related to the general principles of the RASCAL architecture.

6 CHAPTER SIX

REFLECTING ON RASCAL: ANALYSIS, EVALUATION AND DISCUSSION

This chapter presents a general evaluative discussion of the RASCAL framework and its potential as the basis for future collaborative multimedia systems. A critique of the MSpace prototype is also provided along with the presentation of related research to facilitate comparison with similar works.

6.1 General Discussion and Evaluation

This dissertation has presented a framework which addresses the various stages of a collaborative media space's lifecycle. The fundamental tenet behind the research is that complexity must be addressed through simplicity; it is not enough to utilize a collection of objects, each of which performs some collaborative function. Rather, RASCAL is based on combining simple entities in ways that mutually react and adapt to each other, using dynamically flexible combinations (i.e. patterns) of tools and the rules governing their interactions. This separation of behaviour from function provides a scalable and evolutionary approach to media system design through the use of a reflective meta-level architecture. It is also this combinational approach that forms the basis not only for media space structure but also for the behaviours that regulate it.

This section discusses the "pros and cons" of RASCAL as a framework as well as its general topology/architecture. This is augmented by an overview of its potential relative to future system design. And finally, issues and observations arising from work on the MSpace prototype are also offered.

6.1.1 RASCAL: As a Framework of Patterns

Throughout this work, RASCAL has been primarily characterized as a framework for the behavioural management of collaborative media space systems. As such, it offers both a methodology and a set of constructs by which to build a collaborative media system in which behavioural diversity and management are of first-order importance. These constructs, as outlined in Chapter Three, form the basis for components which can combined into patterns of interacting entities. Patterns, as outlined in Chapter Two, offer an approach to help organize and re-use common solutions. They therefore function as building blocks within the boundaries of a framework's overall organization. In traditional structural terms, the two approaches have been distinguished as follows (as taken directly from [GHJV94]):

- *Abstraction*: Design patterns are more abstract than frameworks. While frameworks can be embodied in code, only examples of patterns can be. A strength of frameworks is that they can be written down in programming languages and not only studied, but executed and reused directly. In contrast, design patterns are implemented each time they are used; however, they also offer information as to the intent, trade-offs and consequences of a design.

- *Size and Scope*: Design patterns are smaller architectural elements than frameworks. A typical framework can contain several design patterns but the reverse is never true.
- *Specialization and Applicability*: Design patterns are less specialized than frameworks. While frameworks always have a particular application domain, design patterns can be used in nearly any kind of application. While more specialized design patterns are possible, they do not specifically dictate an application's architecture.

Design patterns can therefore be seen as a related technology to that of frameworks. By providing a complementary way to construct systems, patterns can form part of the "bricks and mortar" that constitute the framework itself. This begs the question that, if design patterns can form part of a system's structure vis-à-vis a structural framework (which specifies what entities exist, how they are connected and so forth), can a behavioural framework, which specifies how entities perform their function and interact with others, be realized from behavioural patterns?

Such a behavioural pattern would borrow from the traditional pattern concept and fit relative to a behavioural framework much like a design pattern would to a structural framework. That is, like design patterns form part of a structural framework, so would behavioural patterns be taken as a constituent part of a behavioural framework (Table 6.1). For example, both types of patterns are smaller than their companion framework and can be reused throughout. However, behavioural patterns would differ from their design counterparts in that their organization, structure and execution more directly affect the behavioural management framework. Overviewed in Table 6.2, a behavioural pattern goes beyond a mechanism to structure implementation and provides an executorial infrastructure which is constituted by the patterns themselves. The abstraction of the behavioural pattern is not the level of structural difficulty it addresses, but the level of user-behaviour it specifies (such as high-level user-centric vs. low-level algorithmic). Size and scope are related to the pattern's abstraction such that complexity is dealt with in terms of its hierarchical definition (via component sub-patterns) rather than by a monolithic approach.

The behavioural framework provides a way to organize different behaviours and apply them at the appropriate level of abstraction, be it relevant to human-oriented collaborative or algorithmic concerns, whichever is correct

Characteristic	Pattern	
	Design	Behavioural
Usage	Common structure-oriented software construction technique	Behaviour-oriented specification and novel software construction technique
Associated Framework	Structural / Implementational	Behavioural management / Implementational
Abstraction	Different structural organizations each of which is realized separately	Different behaviour specifications, policies and management
Size/Scope	Reasonably compact	Varies per task; user-definable
Specialization/ Applicability	Orthogonally reusable across many systems/applications	Varies; reusable across common collaborative contexts

Table 6.1: Design vs. Behavioural Patterns

Characteristic	Pattern to Framework	
	Design	Behavioural
Abstraction	Patterns are more abstract than frameworks; only frameworks are executable.	Pattern abstraction varies and can be composed hierarchically; in the case of RASCAL, they are executed as part of the framework
Size/Scope	Patterns are smaller than frameworks	Patterns are smaller than a framework; in RASCAL, the framework is an "environment" for the execution of patterns; size/scope related to dynamic hierarchical construction
Specialization/ Applicability	Patterns are less specialized than frameworks, which are oriented to a particular domain; patterns are application-independent.	Patterns vary in degree of applicability, but are one typically a prime means of specializing behaviour within the framework.

Table 6.2: Design and Behavioural – Pattern to Framework

for the given scenario. The actual behaviour, however, is provided through the execution of each individual behavioural pattern itself.

As presented previously, the framework provides a behavioural management system based on a reflective meta-level architecture; as such, higher levels can investigate, compute about and induce the appropriate changes in lower-level behaviours. Through the provision of individual behaviours via computationally active components and patterns, the management of said behaviours exemplifies the notion of a *behavioural broker*, in which individual behaviours are "handed out" to meet the "requests" and/or needs of the system, either in terms of resources, user preferences or some other consideration. This functionality, as provided by the behavioural backbone (i.e. the CCS/CMSS vis-à-vis the Proxy Chair) is akin to the broker pattern, which is used to structure distributed software systems (see Chapter Two).

Some of the other patterns mentioned in Chapter Two also offer ways to structure different parts of a RASCAL-based system. Use of publish/subscribe (for provision of multicast data groups), proxy and client/dispatcher/server (as illustrated via DCOM in the MSpace prototype) are two such examples. Metamorphosis, which provides for run-time modifiability, could potentially be used along with reflection to provide a self-modifying system within components (an optional approach, dependent on the underlying technology). The intent and applicability of other patterns, including PAC, view handler, decorator and strategy can seen and realized within the separation of concerns vis-à-vis the use of meta objects and corresponding meta spaces. Examples include the separation of user interface concerns via view handler, PAC and decorator or the isolation and manipulation of computational behaviour logic into "families" of behaviour patterns (effectively realizing the strategy pattern). And finally, reflection, which provides the fundamental essence of the entire framework and its approach to allowing a system to "see itself", is also a pattern.

6.1.1.1 Novelty in Approach

As a framework, RASCAL addresses a number of issues from a unique perspective. These are now highlighted in point form in the interest of brevity.

- A RASCAL-based system is not strictly built out of functional entities. Such entities are mirrored by those which address their behaviour.
- Behavioural components are not implicit parts of the structural objects, but their own active entities (i.e. active objects or actors). They are defined and execute on their own, interacting with other components as necessary to manage the "functionality" offered by them.
- Behavioural components can be combined into composite entities (i.e. patterns). By applying this approach in a recursive manner, components and patterns (taken as components in a larger pattern) can be combined in a hierarchical manner to produce complex and composite behaviour specifications (a collaborative behavioural tree, as outlined in Chapter Three). Each component/pattern has logic which specifies its own specific behaviour or the relationship logic between the constituent patterns/components that are managed by that pattern itself.
- The above separation of concerns is facilitated by use of a meta-level architecture. As a basis for the application of reflection, a component's attributes are reified as separate meta-objects. Therefore, through the use of the active object paradigm, each of these meta objects can actively execute in order to provide a particular attribute's facilities. One such example of this is an entity's behaviour.
- The framework supports use of both computational and implementational reflection. Implementational reflection is used to allow diversity in terms of physical elements such as computational mechanisms including codecs, user interface elements, different applications and so forth. Computational reflection is used primarily by the behavioural backbone and its first-class representation of collaborative behaviours.
- All meta objects, and in particular, all behavioural components, are explicitly first class entities which are explicitly reified and reflectively defined. They can therefore form a reflective tower which can investigate and modify itself (i.e. self-investigative and self-modifying behaviours) to deal with user specifications and/or environment influences (i.e. the ability of the network to provide a specific QoS or the end host to provide an appropriate level of processing or other resources). By employing reflection, there is a natural causal connection between elements used to enforce aspects of a given entity, such as a behaviour or function and its user interface. Doing so allows them to be realized in a causally connected manner such that changes at one location are appropriately sensed in related parts of the system.

Based on these issues, Table 6.3 outlines the typical differences in approach between a RASCAL-based system and those which utilize traditional approaches. As can be seen, the RASCAL approach can be summed up as offering flexibility through modifiability enabled by computability. Comparison at object, function, behaviour and policy levels are provided relative to a set of criteria outlined in Table 6.4.

Entity & Characteristic		Typical Approach	RASCAL Approach
Object	Paradigm	Mixed; predominantly passive	Active actor
	Connectivity	n-ary parent; general graph	Single parent; simplified hierarchy
Function	Role	Domain-specific utility	Domain-specific utility
	Definition	Object-specific internal logic	Object-specific internal logic
	Selection	Usually static	Static and/or dynamic
	Allocation	Usually design time	Design and/or run-time
	Realization	Object – singular entity contains all aspects	Object – unified but explicit separation of concerns
	Adherence	Utilizes inherent/internal control logic	Utilizes explicit/separate control logic
	Connectivity	None – attributes implicit/inherent	Attributes explicit meta objects
	Modification	Code change	Actor replacement
Behaviour	Role	Implicit regulation of function	Explicit regulation of function
	Definition	Object-specific internal logic code	Pattern of interaction rules; arbitrarily complex logic via independent actors
	Selection	Static	Dynamic
	Allocation	Design time, during composition only	Run-time; during composition and throughout execution
	Realization	Inherent and executes as part of encapsulating object	Explicit meta-object which executes independently
	Adherence	Utilizes inherent/internal control logic	Utilizes explicit/separate control logic
	Connectivity	None – implicit/inherent in object	Function or behaviour child, behaviour or policy parent
	Modification	Code change	Actor replacement
Policy	Role	Advisory (passive) regulation of behaviour	Instructional (active executor) regulation of behaviour
	Definition	Inherent in behaviour logic/code	Explicit behaviour actor
	Selection	Static	Dynamic
	Allocation	Design time, during composition only	Run-time; during composition and throughout execution
	Realization	Inherent; embedded within behaviour logic	Explicit; separate logical unit
	Adherence	Executor definition based on design-time/advisory policy	Executor instructed by/interacts with policy (itself an executor)
	Connectivity	Executors different from policies; separate policy/execution "parents"	Executors and policies are orthogonal; only one parent
	Modification	Code change	Actor replacement

Table 6.3: RASCAL vs. Other Approaches

Criterion	Explanation
Paradigm	Whether a passive or active computational entity.
Role	What is the purpose of a given entity.
Definition	What the entity constitutes as a software construct.
Selection	Whether the entity is static (fixed once selected) or dynamic (can be varied/replaced).
Allocation	How a given entity can be brought into being (i.e. instantiated).
Realization	How a given entity is represented within its software definition (explicit vs. implicit).
Adherence	How aspects of a given entity can be enforced.
Connectivity	With what and how can an entity can be connected within the framework.
Modification	How changes to a given entity can be achieved.

Table 6.4: Comparison Criteria

6.1.1.2 Advantages and Benefits

One of the realizations behind RASCAL is that collaboration is a highly fluid and open-ended process. Therefore, the creation of a framework that is open, dynamic and adaptive to both differing technology and potential application were significant concerns.

Based on these elements, the following merits of the framework are briefly overviewed. In particular, it offers the following benefits:

- *object-orientation*: The framework employs the object-oriented paradigm to model participants, media, applications and other items within the media space in a convenient manner. It also works well with the reflective paradigm through support for metaobject protocols.

- *open architecture*: The application of metaobject protocols and implementational reflection endows the architecture with a "glass-box vs. black-box" approach [KP96] [Rao91]. That is, rather than simply integrate components via interfaces with no idea of their inner workings, the internal structure of the architecture and its components are made visible for examination and manipulation via a separate well-defined "meta" interface. This controlled approach to introspection promotes modular, object-oriented and re-useable component-based systems with increased support for customization, extensibility and adaptability while also controlling bad software engineering (implementation) practices.

- *extensibility, modifiability and suitability*: The architecture is extensible in the fact that additional entities can be added to and used within the collaborative process as desired. The open architecture also promotes extensibility by providing a well-defined way to extend the functionality of existing entities and/or modify their current behaviour. Consequently, this also enables diversity in what kinds of systems the framework can support.

- *dynamism and active object orientation*: The framework is defined such that its constructs are non-static, dynamic entities. In attempts to mirror the actual collaborative process, the ability to extend and modify architectural components is not fixed and done during execution.

- *scalability*: The framework scales such that by providing an open implementation, those entities which affect performance can be modified as required to fit their new scope. Furthermore, since the media space is a logical construct, implementation of specific functionality can be done in a manner that suits a particular application of the media space; for example: use of distributed vs.. centralized implementations of entities such as the Proxy Chair (as detailed elsewhere).

- *separation of concerns*: The framework separates media space "execution" from its "coordination" so as to offer cleaner design and implementation alternatives. This separation promotes scalability by promoting a modularity in functionality and reducing the complexity introduced by attempting to do everything "in one spot".

- *policy-driven*: The use of meta levels offers a way to use high-level policy directives to "coordinate" the use of a media space while ensuring coherence via a causal connection between high-level semantic-based policies with low-level "execution" mechanisms.
- *(self-) investigative*: The framework supports the development of a system which can monitor (i.e. investigate) itself at both mechanistic and policy levels.
- *(self-) adaptive*: The framework supports the development of an adaptive system which modify itself at both mechanistic and policy levels.

The above benefits provide a versatile framework that can facilitate a wide range of systems. However, two of them merit additional discussion; these are: scalability and suitability.

Scalability. The scalability of RASCAL can be looked at in two ways: (1) in terms of its physical architecture (i.e. topology); and (2) can the framework address a range of technological and user-oriented abstractions and behavioural requirements. This section addresses the second of these points.

In terms of utility, the ability of RASCAL to scale relates to its application at different levels of abstraction and technical precision. In this sense, RASCAL is scalable in the following ways:

- It can abstract itself to high/low levels detail; that is, the primary concern of implementation can vary between individual behaviours.
- It can build arbitrarily simple or complex hierarchies to address semantic variation; that is; difficult and complex behaviours can be split into combinations of simpler behaviours.
- It does not place the computational burden or cognitive complexity (i.e. the need for a designer, programmer or user to understand) all in one spot.

In relation to the first point, RASCAL's constructs and general methodology can be applied across a range of implementational concerns. That is, different behaviours at different levels can be coded/implemented differently to reflect the primary concern of that behaviour. For example:

- Low-Level Synchronization Behaviour: Concerned mostly with implementation speed, efficiency and real-time response, adjusted relative to the functional component it is dealing with; for example: A/V synchronization.
- High-Level Collaborative Behaviour: Concerned mostly with sufficient intelligence and semantic understanding of user behaviour; for example: is this group of five users working together or is it two groups (of three and two users respectively).

Ultimately, it would be the responsibility of the CMSS to determine whether a given pattern is appropriate for use in a given circumstance.

Suitability. The fitness of any system is related to the domain which it is supposed to support. Because the genesis of RASCAL was the need to facilitate specific behavioural issues (such as mixed media

synchronization) within a collaborative multimedia environment, one of RASCAL's benefits is its ability to support the different kinds of media and interaction that make up the collaborative multimedia domain.

For example, the standard presentational, conversational and interactive media categories are supported as follows:

- Presentational: Utilizes fixed rules which control playback just like other systems.
- Conversational and Interactive: Start with fixed rules (as designed by for the particular scenario or user base) to control the exchange of media data and control events.

This versatility is enabled by the framework's ability to dynamically monitor and re-adjust its behavioural configuration to see if it currently meeting the needs of the entities it is coordinating.

- Presentational: Playback rules can be altered based on the synchronization error rate and/or other parameters measured by the system (such as available network bandwidth or the amount of jitter).
- Conversational and Interactive: Behavioural rules (i.e. synchronization parameters, algorithms) can be dynamically modified by the system as it detects behavioural tendencies and determines how to adjust its resource usage.

6.1.1.3 Issues and Concerns

As with all systems, there are legitimate issues which must be dealt with to ensure a viable realization of the RASCAL framework. This section overviews briefly overviews these concerns and offers commentary on their importance and possible ways to addresses them.

Reflection – Provision and Overhead. As with any "system service", care must be taken in the provision of reflection to avoid inducing difficult or excessive overhead. Readers familiar with the notion of garbage collection and other similar services are well-acquainted with this dilemma of how to provide a useful yet non-obtrusive service.

In terms of provision, the ideal situation is one in which the technologies utilized for the actual system are inherently reflective. By utilizing a natively reflective infrastructure, the resultant systems can easily take advantage of its capabilities without inducing extra computational overhead. Examples include the Prospero CSCW toolkit [Dou96] which is based on Lisp; further details on this system will be offered later in this chapter. For those which are not natively reflective, the implementation of the reflective infrastructure must be done in a way that enables the reflective system itself to be adaptive and responsive to the needs of its users. That is, the reflective subsystem must be adjustable so that different timing and resource constraints of the primary system are not disrupted by use of the subsystem itself. Therefore, like the standard approach to many real-time operating systems, the reflective subsystem must employ a reasonably light-weight and microkernel approach in providing a scalable reflective service. For example, the ability to reflect must not necessarily be encumbered with complex artificial intelligence algorithms embedded within it. Rather, the system must allow

itself to be made as simple or as complex in terms of its evaluatory nature as deemed by the designer and/or user. As an example, MSpace provides a basic infrastructure to support reflective acts across all backbone entities; however, because it embeds such policy logic in its meta level (which is its own actor), the actor can be substituted with one that performs either simple or complex computations.

Non-provisory overhead is also of concern in the responsiveness and latency that can be introduced into the system when it is performing reflective acts. In particular, because RASCAL behaviours are defined in a hierarchical manner using a reflective tower, there is a certain amount of interaction that will be necessary between the different levels of the behaviour's reflective tower. This movement between behavioural levels if called a *Behavioural Cascade*, or *BC*. BCs occur when changes in one of the behavioural levels induces reflective acts throughout other portions of the tower in question. As shown in Figure 6.1, different kinds of cascades are possible, ranging from those which offer a lot of inter-level activity and those that are more limited. Depending on the location of the initiation and the extent to which the rest of the tower is involved, significant overheads could be experienced. Naturally, incurring behavioural cascades very frequently would be detrimental to performance. However, large numbers and frequent BCs would be indicative of an improper design, environmental difficulties or overt sensitivity by the monitoring logic. The desirable situation is a limited BC involving a localized set of meta-levels. Hence, an analogy to virtual memory can be seen in which frequent and extensive cascades can be likened to "behavioural thrashing" akin to memory pages in a demand-paged virtual memory system.

Separation of Concerns. One of the premises of good software engineering is the appropriate separation of concerns. As part of its design, RASCAL explicitly separates domain-specific functionality from its behavioural management. Often referred to as "separating coordination from execution", such a division requires specific effort to provide, especially in terms of the mental shift required to think in such as way (so as to design tools appropriately). Additionally, when the use of existing legacy tools is desired, they would typically require some sort of wrapper to enable this division. An example of this can be seen in MSpace's use

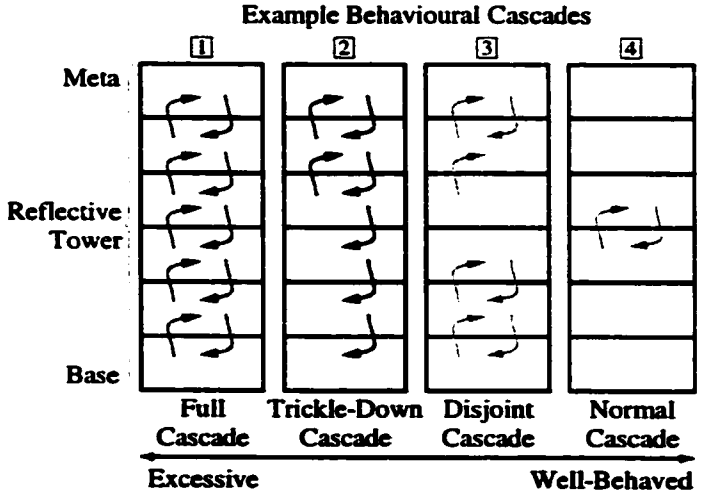


Figure 6.1: Reflective Behavioural Towers and Example Cascades

of OLE and ActiveX controls to provide audio/video functionality.

Pattern Generation and Knowledge Definition. As part of the separation of concerns, the creation of behavioural components and patterns as separate and explicit entities becomes relevant. Therefore, issues related to authoring, accuracy and other logistics need to be addressed. For example, in the MSpace prototype, the finite state machines are authored manually; consequently, the actual behaviour definitions and their specification as rules sets affect the accuracy and usability of the behaviour within the model. The intent is that this procedure should eventually be automated through the use of behavioural editors and/or automatic behaviour recogntion software.

6.1.2 RASCAL: As an Architecture and Topology

In addition to viewing RASCAL as a framework, a generic architectural topology (as outlined in Chapter Three) is proposed for RASCAL-based systems. This section briefly overviews the salient aspects of the proposed approach.

6.1.2.1 Novelty in Approach

The novelty in terms of RASCAL as an architecture is primarily two-fold: (1) as a transparent network-oriented approach like that offered by intelligent networks for telephone services (such as calling waiting and 1-800 service); and (2) as an example of reflective middleware.

Akin to the intelligent network approach, RASCAL attempts to provide a "network coordination service" in which the behavioural management of collaborative activities and tools is done "by the network". That is, rather than embed behaviour logic in end-devices, it is made part of the communication service utilized by the RASCAL-based system (i.e. the behavioural backbone vis-à-vis the CCS/CMSS). While network-based synchronization for multimedia systems has been attempted previously, it has traditionally been addressed as a protocol-oriented solution. RASCAL, on the other hand, addresses the network as a manageable and customizable computational entity which performs behavioural management activities on behalf of the end-user devices. Fortunately, with the increasing interest in programmable networks, such an approach is not only feasible but increasingly popular.

As an example of reflective middleware, a RASCAL media space is illustrated in Figure 6.2. Based on an adaptation of the middleware concept (shown in Chapter Two), a RASCAL media space can be seen to connect the behavioural management middleware component to individual parts of the media space by two interfaces: the standard functional interface and the reflective meta-level interface. Such a configuration allows the middleware not only to provide common functionality (as is the purpose of middleware), but also to provide a standard way of controlling that functionality and its adaptation vis-à-vis reflection.

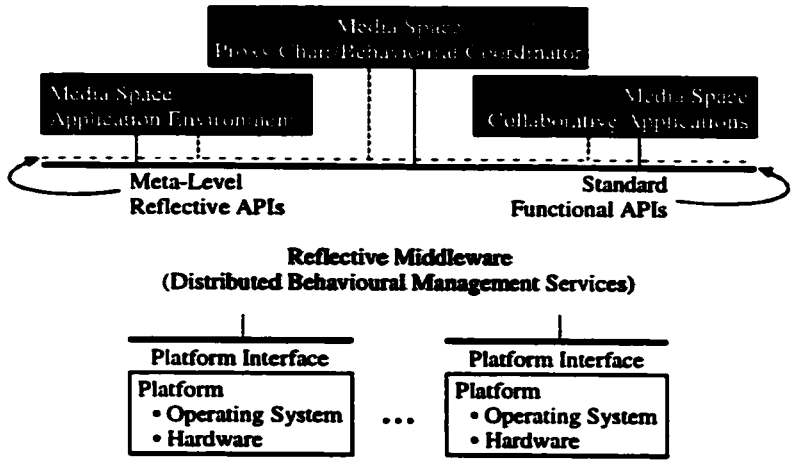


Figure 6.2: MSpace Architecture and Reflective Middleware

6.1.2.2 Advantages and Benefits

By design, the RASCAL topology (as presented in Chapter Three) is rather simplistic. Often, the initial reaction to its centralized approach is one of surprise, given the potential complexity the framework and its systems hope to address. However, this approach is in fact an important advantage to the architecture. Specifically, it is a logically centralized approach, not one which mandates any one approach to its implementation. For example, while RASCAL's services are provided by the Proxy Chair as part of the CCS/CMSS, based on the notion of middleware, distribution of those services is entirely up to the discretion of the system architect. Consequently, the design of RASCAL is implicitly scalable in that it allows this freedom. This issue will be explored further in the upcoming section.

6.1.2.3 Issues and Concerns

Aside from networking issues associated with all multimedia systems (such as QoS and so forth), the most common concern with the RASCAL topology is its centralized approach. While proposed as a potential benefit (see previous section), the centralized approach does require diligence in how it is realized. This section briefly overviews how to possibly deal with such an issue.

A key observation is that the Proxy Chair does not deal with media data; rather it deals with its coordination. Hence, the Proxy deals with control events that describe the flow of multicast data. (as was presented in Chapter Three). Therefore, the potential load on the Proxy Chair is not strictly related to media/application content but rather its use and the precision, granularity and frequency at which events occur in the space. Consequently, this "lightened load" inherently allows a simpler implementation of the Proxy to still be sufficient in terms of the message traffic it would receive.

However, consider the situation in which the amount of computation and/or message traffic causes degradation in service and warrants additional processing capabilities. Because of the middleware and "network service"

approach, the realization of these services within the network (i.e. by the Proxy Chair) allow them (i.e. the Proxy Chair) to be modified in a transparent manner. Therefore, a distributed Proxy could be designed and implemented. Such a Proxy could divide message traffic and computational load in numerous ways (as would be up to the system architect). Examples include having specialized Proxy clusters which deal with audio/video processing and others which deal with static media, such as images and text. Alternatively, Proxies could be allocated on the basis of some network-specific characteristic, such as geographical region (based on network address information). Yet another could involve a set of Proxies, each of which deal with different processing capabilities and which map the management of applications and users based on the computational complexity of their associated behavioural patterns. For example, those which complex patterns over a certain number of levels (perhaps indicating the number of users involved in a certain collaborative grouping) or that contain certain types of processing logic (such as artificial intelligence logic) could be assigned to specific Proxies.

6.1.3 Beyond the Present: Evolutionary Potential

The primary strength of RASCAL is its perspective: the collaborative process and its constituent elements are taken as entities which are dynamically adaptive and require coordinated interaction between them. Media space elements are addressed in an evolutionary and developmental manner rather than as functional specifications of particular applications. Therefore, in some senses, the key to RASCAL's potential is what it does not provide, as much as what it does.

Therefore, this section offers two perspectives for evolution via RASCAL: (1) that based on self-adaptivity; and (2) that based on use of an existing system, like that of MSpace.

6.1.3.1 Self-Adaptivity: Potential, Provision and Application

Consider a discussion-format scenario in which an audio narration (annotation) to some visual aid is also being offered. The narrator may wish to move back-and-forth between the annotation and the discussion with other participants in the space. Therefore, instead of strictly using a voice/picture alignment strategy to control progress (which would typically be the default for the stand-alone annotative case), the user may wish to utilize a voice-analysis meta-level component which has the capability to analyze the intention of the voice-over. That is, the user may wish to use a behavioural pattern that can analyze whether his/her vocalization is being directed to a A/V synchronization pattern (to dynamically start/stop/resume video playback) or whether it is directed to a free-flow audio chat pattern which enables full duplex non-synchronized audio chat between participants. Also, based on aspects such as resource availability, cost and/or security issues, selection of collaborative focus could in fact be denied.

This particular problem (as illustrated above) is how to decide which media and users are related within the collaborative effort. In a real-life setting, physical location and interaction with/clasping of an object defines who is interacting together and how people in that context are behaving. For example, a group of three people sitting at desk drawing on a piece of paper defines a natural collaborative grouping that would be separate from

those drawing on the whiteboard at the other end of the room. Within a logical space, however, defining who is interacting with who and what facilities they are using is less obvious. Furthermore, if a person at the desk stood up and went over to the whiteboard, the shift in collaborative focus would be obvious. Within a logical environment, however, movement between collaborative groups somehow needs to be recognized and facilitated.

This need to go beyond simple behavioural management and provide for "intelligence" in the application of behavioural management is shown in Figure 6.3. The ultimate goal of this hierarchy is to provide appropriate behavioural control based on user's collaborative focus is called *Semantically-Adaptive Synchrony*, or *SAS*. The principle of SAS is that the behavioural management (i.e. synchrony) required in a particular situation needs to adapt itself based on the semantics of that collaborative effort. Recognition of such a "situation" along with its semantics is a dynamic, complex but not necessarily conclusive/determinate process. Being very context-dependent, SAS requires an intelligent approach to managing the space, its entities and their behaviours. Therefore, *Intelligent Behavioural Management* (*IBM*) is required by a system in order to recognize known well-known user behaviour patterns and create new ones as they become commonly used in the system. This intelligent learning approach is based on a dynamically modifiable structural and behavioural organization that can compute about itself for the purpose of adapting itself to meeting the new configuration(s) that arise from changing usage. Through the use of reflection, this intelligence can be realized via an arbitrary meta-level architecture, being as simple or complex as the designer wishes to make it. Such an approach to synchronization, called *Reflective Synchrony*, is a means of providing intelligent behavioural management though the use of reifiable behaviour models which can tailor their synchronization specifications "on the fly". Taken further, Reflective Synchrony allows for *INtelligently Self-Adaptive and INvestigative (INSAIN)* behaviour management (Figure 6.4, see also Figure 3.4) via a reflective tower that can be abstracted to an arbitrary level of "intelligence". This offers the potential of dealing with behavioural semantics, pattern recognition and so forth at higher levels in the meta-hierarchy that can adapt over time using the appropriate technologies as they become available.

Within Reflective Synchrony, two different models for the application of reflection can be envisioned. Abbreviated as *PARS*, the *Passive/Aggressive Reflective Synchrony* models are shown in Figure 6.5. The

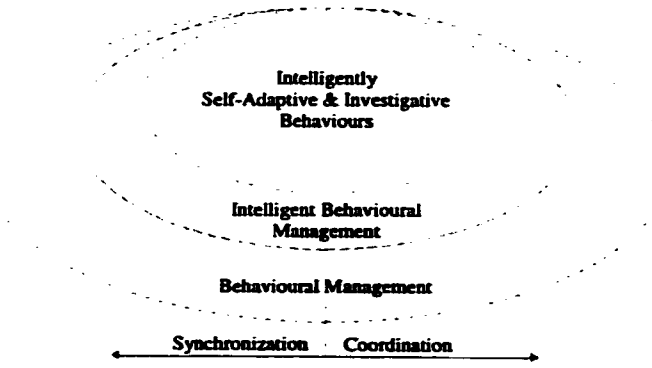


Figure 6.3: Extended Behavioural Management Hierarchy

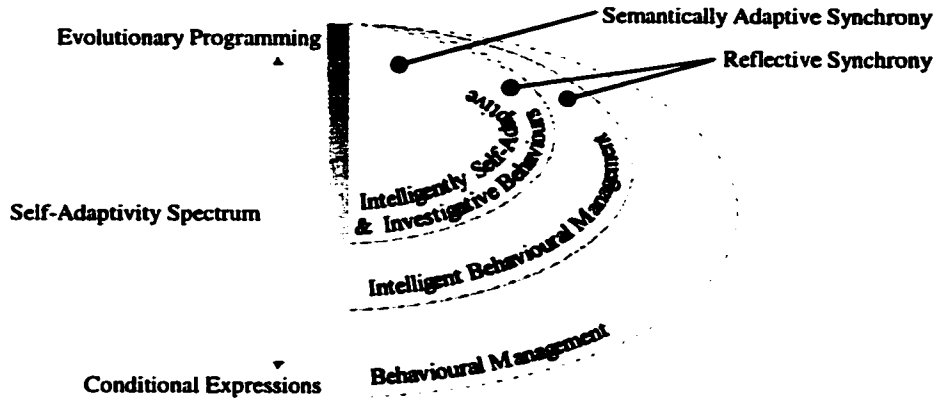


Figure 6.4: Self-Adaptivity and the Extended Behavioural Management Hierarchy

Passive approach is one in which a meta-level performs its meta-level computations in response to a request from its base level. The Aggressive approach is that initiated by the meta-level; that is, the meta-level behavioural pattern continually computes about its domain (the base level) and pre-emptively intervenes as it deems appropriate. The RASCAL framework does not dictate which kind of approach to use; in fact, either is possible – both separately and in combination within the system. The appropriate choice is left to the system designer.

Consequently, different behavioural patterns can be seen to offer specific *Quality of Interaction (QOI)*. These differing behaviours can offer different levels of accuracy, synchronization support and the like depending on need of the collaborative scenario, user preferences or system resource levels. Example media characteristics that could be utilized to provide variation in object QOI include: reduced input (i.e. mouse) sensitivity, progressive resolution coding; colour vs. b/w rendering and/or different frame rates.

6.1.3.2 Building On Experience: Exploring MSpace

This section briefly overviews the experience garnered from the MSpace conceptual prototype. Lessons learned from the implementation itself and its potential for future implementations and simulation studies are also mentioned.

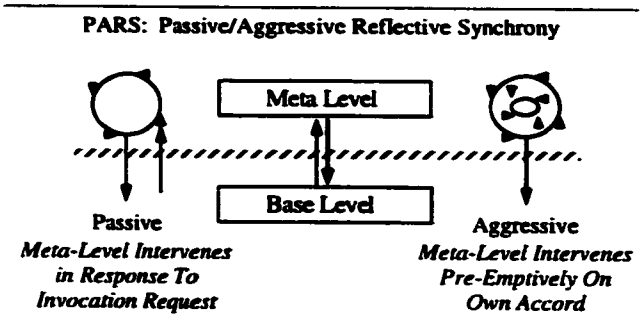


Figure 6.5: Initiating A Reflective Act

6.1.3.2.1 The MSpace Conceptual Prototype: Lessons Learned

Detailed in earlier chapters, the development of MSpace was based on the integration of a variety of PC-based technologies and development tools. While independently valuable technologies, their particular combination proved more difficult than intended for the desired implementation. In particular, the following aspects proved problematic:

OLE Legacy Technology: Unfortunately, a significant deficiency of OLE in terms of real-time applications is its use of static server representations when an object is not the focus of user input (known as "in-place active"). Specifically, there is no standard means to handle modern real-time "continuous sourced" entities (e.g. video) and their user interface *in all cases*. This issue of "user interface integration" and "user interface liveness" (see Chapter Two) effectively means that when the video object is not in collaborative focus, obtuse (and typically non-performant) means need to be used to update the user interface.

D/COM Complexity. While a viable example of middleware technology, many of the aspects of COM and DCOM are not particularly well-suited to real-time applications. In particular, simple issues such as the "heaviness" of its implementation and its communication model (such as lack of true multicasting support) are too difficult and awkward to work around. Simple issues such as *data associativity* (i.e. how to relate data to the appropriate application tool) can be extremely frustrating in real implementation terms. Additionally, the ability to programmatically manage ***event manipulation*** (i.e. how to notice, intercept, respond to and effect application events and changes in data programmatically) is only partially supported. Some of these issues are supposed to be addressed in the upcoming COM+ model, which will first be available as part of the new Windows 2000 architecture. However, for purposes of MSpace's development, they were not available.

Closed Technologies. Many of the technologies (such as ObjecTime and MFC) offer mechanisms to expand and modify select elements of their implementation and how they provide services to software developed using them. In the case of MSpace, however, MFC's support for OLE and DCOM along with ObjecTime's ability to programmatically control actor placement, definition and communication, were examples of when useful implementation details within the tools were "off limits" and could not be used in the desired manner. Effectively, the difficulty emerged from trying to create an "open system" out of ones that were inherently "closed".

"Too Many Masters": Several of different technologies utilized within MSpace had one additional problematic aspect about them: they were all frameworks that assumed control over program execution. Consequently, the integration of these frameworks (such as MFC, DCOM and ObjecTime), resulted in multiple "masters", each of which had the perspective that it was "in charge" of system execution. Consequently, providing a way to integrate them proved extremely difficult and in some cases, was unsupported.

6.1.3.2.2 Future Implementation Possibilities

The MSpace conceptual prototype is itself only an example of the kinds of systems that can be realized. For example, through its use of collaborative object applications (via OLE components), the actual MSpace

prototype can take on different characteristics and functionality. Therefore, in order to illustrate how MSpace was built and how it can be used as a basis for similar systems, this section provides a brief overview of the classes used to construct it. As such, these classes (some which are shown in Figure 6.6) are a realization of the generic RASCAL class framework outlined previously.

To build a MSpace media space, the designer first instantiates a top-level containing instance of the RASCALMediaSpace class. Instantiation of this object maps to the creation of the media space; its destruction constitutes the removal of the media space as an environment in which collaboration can take place. For any collaborative system, there will be only a single instance of the RASCALMediaSpace.

Within the media space, all entities are derived from the top-level base class known as RASCALBase. This entity provides the core functionality required by all entities: reflectivity. This support for reflection is provided by another top-level entity called RBS_ReflectionService. Because the current technologies used to implement RASCAL are not themselves reflective, support for reflection must be facilitated externally. It is the intended role of this service to provide such support for entities within the space. To do so, access is inherited

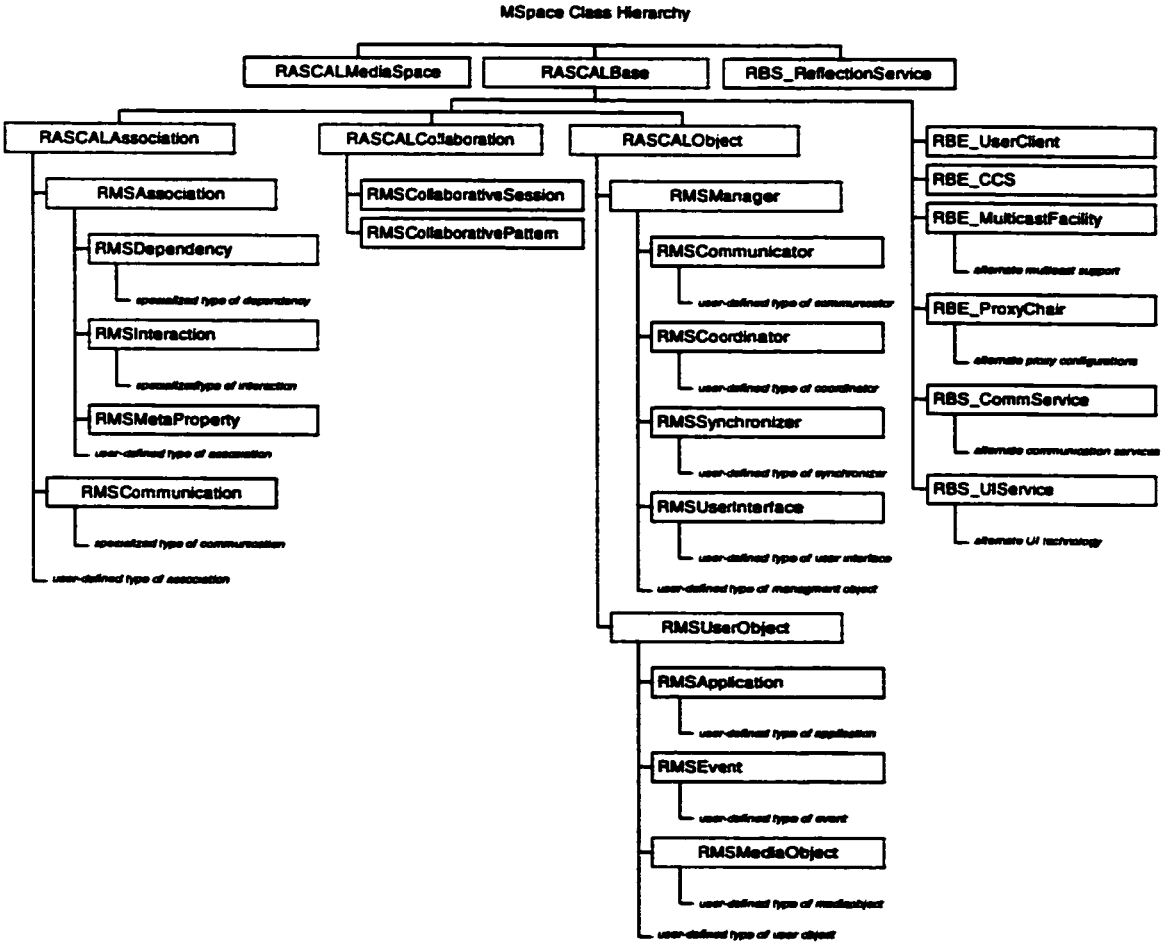


Figure 6.6: Illustrating Some of the MSpace Classes

though the RASCALBase class.

From the RASCALBase class, there are three abstract RASCAL classes and a series of RBE_ and RBS_ classes defined (as a reminder, these prefixes are explained in Chapter Five). The abstract classes are the classes which will be used by most media space designers to build their system. Entities built around these abstract classes and their derivatives form parts of the media space which the users use within their collaborative activities. The remaining RB*x*_ classes are intended to provide environment-wide entities and services which enable operation of the space. However, in order to facilitate flexibility in the configuration of the media space's support entities and services, these classes also inherit the ability to reflect though their derivation from the RASCALBase class. The potential utility of such an approach is the ability to dynamically reconfigure the media space environment vis-à-vis reflection. Such a reconfiguration could be based on a user interface to enable manual setup. Alternatively, complex decision logic (i.e. "intelligence") could be embedded in an entity's meta space which is accessed through the use of reflection. Examples include changing the behaviour of the multicast facility or change the realization of the Proxy Chair from a centralized implementation to a distributed one. Dynamic changes in UI services for a given entity could also be based on such an approach.

The three primary abstract classes of concern to the average RASCAL designer are RASCALAssocation, RASCALCollaboration and RASCALObject. These classes effectively form categories from which other abstract and concrete classes can be defined. Each of these classes will now be discussed in order of their use in creating a realizable media space.

In order to utilize the media space, users participate in a collaborative session which is an instance of the RMSCollaborativeSession class. A user may theoretically belong to many different collaborative sessions, although this is equivalent being "in more than one place at one time". Hence, while a user can be a member of any number of collaborative sessions, he/she will be interactively active in only one of them at any point in time (this is not a limitation of the framework but of physical reality). On the other hand, a user could conceivably launch a media object into a session which would remain active (and producing output to the other users) while he/she is interacting in another session. This would give the appearance of simultaneous activity within two separate sessions. Sessions are created within the media space and form the logical entities in which collaborative activities occur.

Within a session, a user collaborates through the manipulation of collaborative objects derived from RMSObject. These user-level objects represent the entities that participants in the collaborative session use to perform their work. Examples could include user-defined applications (RMSApplication), events (RMSEvent) or type of medium (RMSMediaObject). RMSMediaObject is itself an abstract class that would be subclassed to provide the specific functionality of a particular media type. For example, it could be subclassed in the following manner to provide support for different types of codecs (Figure 6.7).

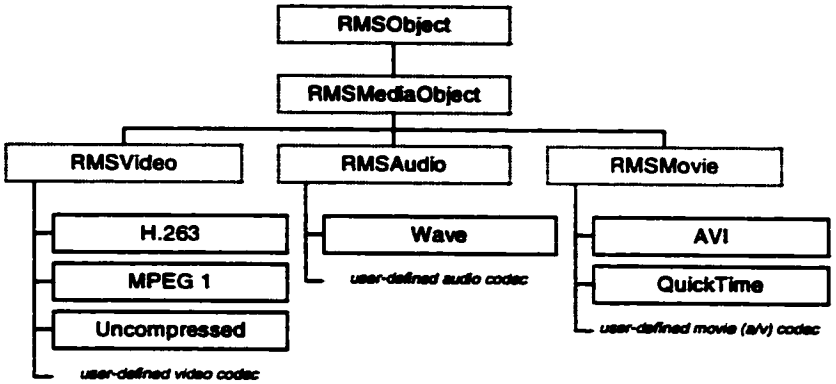


Figure 6.7: Example of Possible Media Object Hierarchy

It would also be possible to simply consolidate support for media objects under the RMSApplication class. While such could still be done, especially for the collaboration which involves the use of applications for editing media, media objects were provided as a separate class for the following basic reasons:

- Users may wish to collaborate using applications that are not strictly media-based.
- Users may wish to collaborate using media that tend not to be application-centric; for example, live audio/video conferencing could be facilitated in a light weight fashion by employing separate video and audio "applets" rather than strictly using a video conferencing application. Alternatively, raw data feeds (video, audio or images) with no controls could also be envisioned.

Conceivably, either could be done according to the choice of the media space designer. Therefore, such an approach was seen to be more flexible and generally a cleaner separation of object types. Type of synchronization or media mode used for media are not mentioned at this point as such information is a meta property of the medium itself and can be associated with the object at a later date (via dynamic property binding via its metaspace).

The RMSObjects are entities which are manipulated as part of the collaborative process. How they are manipulated, however, is demarcated and controlled through the use of instances of classes derived from RASCALAssociation, or more particularly, RMSAssociation. These classes define the constraints between objects, be it specific media, applications or events that the user has defined and associated with a particular object type. For example, a RMSEvent could be defined and associated via an existence dependency (possible RMSDependency subclass) with a particular application. Hence when that application comes into existence in the media space, the event would be fired which could itself satisfy an existence dependency for another application in the space, allowing it to proceed. Hence, you can define complex associations of different types between objects. These associations form the way objects are intended to interact and therefore form part of the behaviour patterns which govern the actions within the media space. These behavioural patterns

(RMSCollaborativePattern) are designed using the notion of aggregation in which more complex behaviours are formed by combining simpler behavioural patterns together (using some sort of association).

To coordinate the space, the actual input actions generated by the users need to conform to the behavioural patterns which govern the acceptable sequence and timeliness of events in the space. This adherence is provided by use of RMSManager objects. These entities effectively police the rules outlined by the behavioural patterns/components. In particular, the RMSCoordinator and RMSSynchronizer manager objects are responsible for different levels of coordination and management within the space. Furthermore, they can be subclassed to specialize in their management of certain types of behaviour, including media mode, granularity, limitations of undo/reverse-ability (e.g. internal dependency within MPEG frames) in terms of associated codecs, etc., and be related dynamically to their objects of concern through an RMSMetaProperty association.

RMSManagers are instantiated as meta objects and linked via RMSMetaProperty associations to the objects on whose behalf they are delegated functionality. For example, an audio media object would have its own UI and synchronizer; so would a video object; and they would share a coordinator that would interpret and monitor their behaviour relative to an A/V lip-synch synchronization behavioural pattern.

As part of building such an environment, a client application at a workstation represents the "point of presence" used by each participant to connect to the space. Via this application, users would create and/or join collaborative sessions, while within each session, a user would insert and/or manipulate objects that are used as the basis of collaboration. Table 6.5 illustrates the relationship between elements of the RASCAL Object Hierarchy and components of the system.

- *Instantiated*: Class is used directly "as is".
- *Jointly instantiated*: Entity is shared across the implementations of the backbone CCS and its front-end.
- *Derivation*: Specified class is used as a root for more specialized classes.
- *Correspondence*: Neither the class nor its derivations are directly used at this point; rather, a "placeholder" which corresponds to it is used in its place.

It should be noted that the object hierarchy is defined as a structuring framework rather than strictly a series of building blocks from which all components in the space must be explicitly derived from its class definitions. While the actual definition of the hierarchy is done though the ObjecTime toolset, objects may or may not be built directly using that hierarchy. In many cases system entities will be represented in multiple locations as "variations on a theme".

For example, consider the media space itself. It is defined as a top-level class whose instantiation functions as an existential entity for when you build a collaborative media space system. When the system is running this entity is instantiated and in terms of the ObjecTime CCS/CMSS backbone, it forms a structural container for the rest of the media space model. However, in terms of the rest of the system, it is not directly instantiated but

System Entity	Representative MSpace Classes	Location and Correspondence of Entity		
		CCS/CMSS Backbone	CCS Front-end	User Client Point of Presence Application
Media Space	RASCALMediaSpace	Instantiated	—	Set of client applications
Collaborative Session	RMSCollaborativeSession	Instantiated	Correspondence	Correspondence, instantiation or derivation
Collaborative Object	RMSUserObject-derived classes	Instantiated	Correspondence	Correspondence, instantiation or derivation
Collaborative Behaviour	RMSCollaborative Pattern	Instantiated	—	—
Object Relationships	RASCALAssocation-derived classes	Instantiated	—	—
Management Objects	RMSManager-derived classes	Instantiated	—	—
CCS	RBE_CCS	Jointly Instantiated		—
Proxy Chair	RBE_ProxyChair	Instantiated	—	—
Multicast Facility	RBE_MulticastFacility	Jointly Instantiated; Functionality is delegated to front-end		—
User Client Workstation	RBE_UserClient	—	—	Correspondence; where the client application executes
		ObjecTime & C++	<All>	C++/MFC/ATL, OLE/DCOM, ActiveX
		Corresponding Technologies used in MSpace		

Table 6.5: Entity Correspondence Summary

corresponds to the collection of user client workstations, the client applications they are running and the sessions/objects they are manipulating.

The collaborative sessions and the collaborative objects are slightly different in that they are manifested at each stage of the system, but not necessarily in the same manner. Within the CCS/CMSS backbone based on ObjecTime, each of these entities is built explicitly from their representative RASCAL class. Entities within the backbone form the computational rather than user-interactive part of system. As defined in ObjecTime, they could be used to generate end-user objects that the user interacts with. However, ObjecTime is not particularly adept at providing a multimedia user interface for the applications it generates; therefore, different technologies and techniques were used to provide the actual client application and collaborative object tools used within it. For the prototype, these entities were built using Windows NT-based operating system constructs and development tools. They are built to "plug in" to the framework in which the instantiated backbone variant of the object is the one which is "managed" while the client-application variant provides the "end-user experience". The CCS front-end provides the necessary translation between the backbone and the client end; this allows the client end to change and isolates the changes necessary to interface different client

technologies to the backbone. In effect, the behavioural backbone is isolated from client UI dependencies and each functions as a proxy for the other's functionality. It also provides a suitable place to put a specific management/monitoring interface for the CCS/CMSS backbone if one is desired.

6.1.3.2.3 Simulation Studies

Because of ObjecTime's ability to function as both a simulation and code generation environment, MSpace entities that are built within ObjecTime can also developed and tested using its simulation facilities. By supporting internally built test harnesses as a means to validate its models, ObjecTime offers the ability to independently simulate, test and analyze any of the prototype's ObjecTime-based components, including the reflection service and the behavioural pattern mechanism. Consequently, behavioural patterns for collaborative scenarios and different synchronization configurations could be built and tested before using them in an actual collaborative setting. The ability to check for excessive behavioural cascading could also be investigated. One challenge, of course, would be the generation of realistic interaction input models to drive the behavioural models.

Therefore, such a configuration can be thought of as a *collaborative simulation environment* (CSE) by which different media space components (such as behavioural patterns and the like) can be tested via simulation before generating an actual run-time system.

6.2 Related Research and Systems

Collaborative space research has traditionally revolved around elements of collaboration arising from process, sociological and technological perspectives: (1) specific application environments [AEH88] [Eri94]; (2) human-computer interaction [GG95] [LL90]; and (3) communication and/or application protocols [Yav92] [AY96]. Additionally, multimedia research, especially with respect to synchronization and networked architectures, often take very specific viewpoint, ranging from authoring mechanisms to QoS. Still others focus on specific implementation, algorithmic or protocols issues. While not necessarily focusing on any of these particular issues in detail, this dissertation does not preclude any of these efforts. In fact, it can be viewed as a potential a way to organize, relate and integrate different approaches (including user interface technologies, applications and protocols) as required. Consequently, specific research efforts that bear relevance and/or some comparison are now overviewed.

6.2.1 Multimedia Collaboration

While there have been a significant amount of work done in multimedia and collaborative systems, much of it has had a specific focus in terms of one issue or the other. For example, traditional desktop conferencing systems like MMCONF [CMB+90] and Rapport [AEH88] focused mostly on the use of audio/video. Rather than address a plethora of specific systems which focus either on multimedia or collaboration per se, a range of

efforts that illustrate their combination are overviewed (and summarized in Figure 6.8). These include: multimodal environments, collaborative virtual environments and (media-capable) CSCW tools.

Multimodal environments (MMEs) are a family of systems capable of establishing creative, multimodal user interaction by exhibiting real-time adaptive behaviour [CF99]. Users typically communicate by means of full-body movement and vocalization (e.g. talking, singing or playing), and receive real-time feedback from the environment in terms of sound, music, visual media and actuators. MMEs therefore form a kind of *augmented reality*, one in which live objects exist in a synthetic environment. The interaction between computational entities and human gesturing is possible using bio-feedback methods in which MMEs act as agents that observe the user, extract "gesture gestalts" and change their state over time. As part of their research in this area, [CF99] presents four "artistic" applications based on their MME architecture with an emphasis on the realization of artificial emotions:

- *HARP/Vscope*: This application tracks full-body human movement using on-body, wireless sensors, for gesture recognition and real-time control of computer-generated music and animation.
- *SoundCage Interactive Music Machine*: IMM uses a set of spatial sensors are used in a "cage" to overall, full-body human movement without the need for any on-body devices or constraints.
- *HARP/DanceWeb*: Ultrasound sensors are used to acquire human movements, either in stand-alone installations or integrated with the SoundCage IMM.
- *Theatrical and Museal Machine*: TMM consists of one or more semi-autonomous mobile robots which can perform tasks such as a museum tour guide, or robot-actor on stage at cultural events and art installations. The system is able to exhibit an "emotional state" through sound, visual media and its movements.

Similarly, the *Artificial Life Interactive Video Environment* (or *ALIVE*) [MBD+95] is an augmented reality system in which people can interact with virtual creatures without being encumbered by traditional VR equipment, such as headsets, goggles and sensors. Based on a "magic mirror" metaphor, a person using the ALIVE space sees his/her own image on a large-screen TV as if in a mirror. Animated characters are added along side the user's own image (via video compositing) in the reflected world, enabled through and managed by autonomous intelligent agents which control their behaviour and interaction with the user. Using a vision-based tracking system to extract the user's bodily position and gesture information, the autonomous characters use this information along with their own motivations (i.e. behavioural logic and/or policy) to act in believable and entertaining ways. Therefore, the user can interact (and hence potentially "collaborate") with various agents via their representations within the world. The end result is a system which illustrates the interaction between synthetic and live characters found in such films as "Who Framed Roger Rabbit" [ILM88].

An interesting approach to teleconferencing is illustrated via the TELEPORT project [GAB99]. In this research, users are situated in an office which is "virtually extended" using an augmented reality interface. The intent is to manipulate the visuals in a specially designed environment which manipulates projections of a

remote office to make the user perceive the remote space as physically attached to their own. This so called "immersive copresence" illustrates a novel user interface approach similar in intent to the composite effects of the ALIVE project. While the potential to collaborate is quite motivating and addresses many HCI issues, there is no real emphasis on the management of the collaborative behaviours provided.

Collaborative Virtual Environments (or CVEs) are a form of virtual reality (VR) systems whose focus is on enabling collaboration between multiple participants within the shared VR space. As opposed to traditional virtual reality systems which focus on simulation of a physical environment, CVE's address the interaction between people and are sometimes called "social virtual reality" systems [MER00].

One approach to CVE's is the proposed middleware standard for multi-user virtual worlds called Open Community. It is based on the SPLINE (Scalable Platform for Large Interactive Networked Environments) technology originally developed by Mitsubishi in relation to their Diamond Park DVE prototype [ABH+95]. PlaceHolder [LST94] was an early research initiative which explored a media-based paradigm for narrative action in virtual environments. Primarily focused on non-technical issues, it used a navigational technique called "placemarks". These markers, either graphical (iconic/pictorial) or aural (so called "voicemarks"), would then serve as "signposts" for people as they traversed the shared space. Other approaches include projects such as MASSIVE-2 [CRG97], DIVE [Hag96] and NPSNET [MZP+94]. These systems are highly specialized and highly powerful immersive 3D environments which focus on varying issues related to development of purpose-specific, large scale and/or physically realistic DVE's. While it beyond the scope or intent of this work to address the issues involved in such systems, they illustrate the different kinds of interface techniques that can be utilized by a collaborative framework.

Certain CSCW tools also offer collaborative multimedia potential. These include systems such as JETS [SOG98], its successor Jasmine [SSGS00], JAMM – Java Applets Made Multiuser [BSSS97] and Habanero [CGJ+98]. Each of these efforts are Java-based telecollaboration systems which aim to facilitate Internet-based collaboration through the sharing of Java applets. Therefore, the functionality of these systems can range from video conferencing to shared text editing and the like, offering a way to integrate different functionalities in an "application suite"-like approach. The first three projects aim to share applets/applications via the Internet using a standard web browser, while Habanero offers alternative approach, using so-called "Hablets" which require a proprietary browser. Habanero also supports the development of collaborative environments offering a framework to create shared applications, either from scratch or by altering existing single-user applications. In each, integration and sharing of functionality is the primary concern in terms of facilitating collaboration. There is no support for addressing specific behavioural concerns in a wider sense.

Non-Java based approaches to CSCW include projects such as GroupKit [GR98], which provides an emphasis on group management, and MBONE [Eri94], which concentrates on multicast issues. GroupKit does not specifically address multimedia concerns, nor does it specifically deal with behavioural management. Likewise, functional integration is not addressed since the premise of GroupKit is the building of custom applications; in contrast, while MBONE offers a useful suite of tools, they are separate and disjoint applications

that are not easy to use together in parallel without fairly extensive user-level management. Prospero [Dou95a] [Dou95b] [Dou95c] [Dou96] is an interesting approach that relates to this dissertation not in that it provides an integration or multimedia environment, but in that it is a reflective CSCW toolkit. It offers a "build-from-scratch" approach to CSCW development and focuses on constructional vs. integrational flexibility. Unfortunately, because it is based on the Lisp language, Prospero realizes its reflective nature from the fact that Lisp itself is reflective. Another work, the Introspect framework by Tolone [Tol96], is a meta-level model which allows users to specify "settings" for "collaborative contexts". As a meta-level framework, it includes a collaborative support architecture (using a collection of web applets) and a collaborative specification environment (which specifies how collaborative tasks are organized). Therefore, while the use of reflection in CSCW has been proposed before, its use has been primarily as a toolkit design mechanism or high-level collaborative task organization. In particular, its application to multimedia systems and this work's emphasis on behavioural management and definition were not addressed in these two specific instances.

The primary difficulty with the above efforts is that collaboration is either approached in terms of: (1) a specific program's functionality; or (2) how well the elements of a system and/or its user interface suit the needs of the collaboration and its participants. Consequently, behaviour and management are often either not addressed or embedded within a given application. Additionally, collaborative utility is limited by the use of specific tools in a particular context because the collaborative utility is "wrapped up" in the design and complexity of the chosen tool set. A similar viewpoint is offered by [MBK+99] which outlines work towards multimedia networking middleware based on combining the best from several multimedia toolkits into one. However, by virtue of reusing aspects from previous toolkits, the resultant product is again constrained by the designers' choices.

Therefore, more flexibility is offering by regarding these user interface technologies as possible meta-level extensions or UI options which can be integrated with an underlying behavioural management framework like RASCAL. The significance of MMEs is their integration of synthetic computational entities (including media) as active objects which interact with human participants according to some behavioural specification. The focus of most MME work, however, is their user interface and use of agents. The focus of CVEs has been their emphasis on the realization of 3D virtual reality environments while many of the mentioned CSCW tools tend to only have a strong functional integration approach (in addition to a heavy Java-bias). However, each approach can be seen to be a subset of a media space in which the actual user interface can be considered as a meta-level issue (see Figure 6.8). In a sense, the media space is in fact a logical "meta space" which encompasses different kinds of user interaction paradigms independently of the support required from functional and behavioural frameworks like RASCAL.

6.2.2 Synchronization and Coordination

As evidenced by [IEE96], there has been a large amount of work carried out in the area of multimedia synchronization over the last decade. Such research has covered various approaches including specification

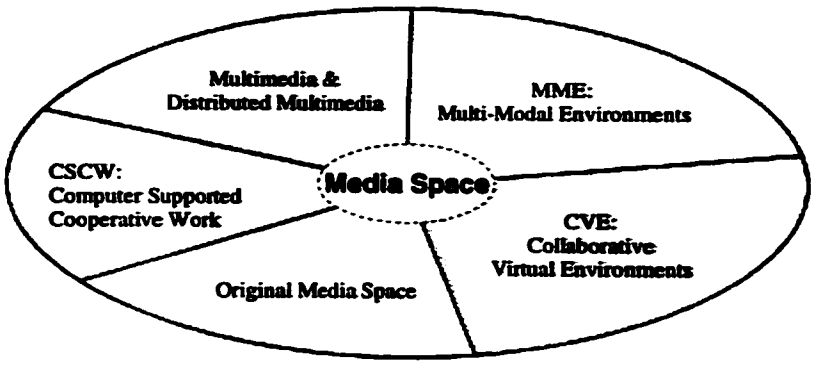


Figure 6.8: Media Space Diversity

techniques, synchronization algorithms and mechanisms, network and processing issues as well as numerous specific systems. Most of this work has primarily been considered in one of the traditional presentational, conversational or interactive domains; therefore, while applicable in parts to the diverse requirements of a collaborative media space, they do not suite the overall breadth and diversity of such an environment.

Numerous approaches [LG90] [LG91] [Gib91] [GDT91] [AH91] [BCC+92] [CCGH92] [LK92] [RR92] [LKG93a] [LKG93b] [DNNR93] [WQG94] [Ack94a] [Ack94b] [Rob95] [SKD96] [Rob97] to media synchronization within presentational systems can be found in the literature. In general, they can be characterized as follows:

- *layered construction*: Synchronization is addressed at multiple stages, using different entities and mechanisms to correct asynchrony as it becomes noticeable (e.g., the network and playback levels, within and/or between streams, etc.).
- *object abstraction*: Media data and system components are modeled as independent but interacting objects. Several use the "active object" (*aka* "actor") paradigm.
- *event synchronization enabled through scheduling*: Coarse-grain event-level synchronization is facilitated by using scheduled media object playback times.
- *continuous synchronization enabled through fine-grain temporal intervals*: Fine-grain continuous synchronization is achieved through the division of a media object into a series of small temporal sub-divisions which are individually aligned to their correct playback time.
- *interactivity via pre-computable paths and choice points*: Some systems support user control of playback and/or the ability to "move around" within the presentation. Those that do utilize some sort of "choice point" at which multiple paths through the presentation can be taken. Because they are known in advance, the path and its requirements in terms of synchronization and other resource requirements can be pre-computed prior to their navigation.

Conversational systems, on the other hand, tend to rely more on protocol-based techniques. Examples include the MBONE [Eri94] audio tools *ivs* and *vat* which use the RTP [SCFJ96] protocol. In these systems, the

synchronization mechanism functions as a protocol engine ensuring that the data conforms to the protocol. This is in contrast to presentational systems in which the mechanism ensures the data conforms to an external specification (i.e. the presentation's scenario timeline).

The deficiency with most presentational systems, in terms of a real-time media space, is their reliance on scheduling and the pre-computation involved. In particular, such scheduling is not always possible within a real-time collaborative environment. Conversely, the protocol approach, while more flexible in terms of real-time requirements, is based on the notion that a set of data exchange rules is the main issue. In some systems, this approach degenerates to basic floor control [DG97], yielding "message sequence chart"-like solutions ranging from the very general to the very specific and complex. In contrast, this work addresses the need for a framework and methodology which can be used to integrate these techniques; this enables the realization of a versatile and enabling infrastructure for user-level collaborative applications that considers multiple types of behavioural needs simultaneously.

6.2.3 Multimedia Middleware

In an attempt to bridge the functionally rich but difficult and error-prone programming interfaces offered by modern networked operating systems, multimedia middleware attempts to layer useful abstractions and support services between the OS and multimedia applications. Two such projects are outlined here: GOPI and Agilos.

The Generic Object Platform Infrastructure (or GOPI) [Cou99] from Lancaster attempts to provide configurability, performance and predictability within a standard operating system environment. Basic "core" services (including thread, communication, buffer and other components) are implemented as independently replaceable modules whose services are offered to upper layers using an appropriate "personality" subsystem (such as the CORBA interface with multimedia extensions). This core subsystem maps standard OS services to those appropriate for multimedia while the personality subsystem allows GOPI to present its 'best face' to the appropriate application layer.

Agilos [LN99a] [LN99b] is a middleware control architecture designed to provide QoS services to assist application-aware adaptations; that is, adaptation mechanisms tuned to the performance goals and specific functionalities of an application. The objective of Agilos (which is short for "Agile QoS"), is to control the adaptation process within the application so as to maximize user satisfaction with the system's performance under any resource condition. Agilos is designed as a three-tier architecture:

- Tier 1: The lowest tier utilizes application-neutral mechanisms which maintain tight relationships with individual types of resources, reacting to changes in their availability.
- Tier 2: Application-specific "configurators" are responsible for making decisions on when and what adaptive mechanisms are to be invoked in a client-server application, based on on-the-fly user preferences and application-specific rules.

- Tier 3: Deals with group-level behavior adaptation, so that dynamic reconfigurations of client-server mappings are possible and tuned to application.

Hence, Agilos can be seen as employing a meta-level approach to QoS management but does not offer the ability to modify itself in an orthogonal manner. GOPI, on the other hand, offers a bit more support for self-configuration thought its "personality" subsystem, but offers more of a programming interface than an adaptation service. Therefore, both of these projects are not quite at the same level as the work found in [BCC+99] [BCRP98] [CBC98] which introduces a *reflective middleware* approach to QoS management. In any event, multimedia middleware systems can be seen as primarily dealing with the issues of QoS management, adaptation and abstraction. However, the breadth of behavioural management support offered by RASCAL has not yet been addressed.

6.2.4 Reflection and Reflective Systems

The application of reflection has seen increased acceptance across a range of technical domains in recent years. Moving beyond applications in linguistic and AI fields, reflection and meta-level architectures are increasingly being exploited in distributed and real-time computing. Additionally, these techniques are also being used as a way to provide flexibility not only at a machine level, but to meet the needs of human users.

6.2.4.1 Distributed Computing

One significant systems area that has received considerable attention from the reflection community is that of distributed computing. Seen as a way to mediate between complexity and flexibility, reflection has been used to separate the "what" from the "where" within a variety of distributed systems. Two such examples are: (1) the distributed object architecture called Tj/CodA; and (2) the Muse distributed object model and related operating systems Muse and Aperios.

Distributed Objects. [McA95] presents a meta-level architecture for transparently adding distributed behaviors to objects without substantially affecting their base-level. The motivation for this work is to provide an environment which allows designers to explore and experiment with various computational and structural models for application objects. The main feature is the separation of the base-level application code from the meta-level object distribution code. A platform called Tj was developed to describe distributed object behavior and is built on top of CodA [McA93], a general meta-level architecture which provides access to the implementation of an object's basic computational elements (e.g. state, message passing, queuing, method execution). Tj's distributed object model consists of mechanisms to deal with object spaces, machines, topologies, remote references, object marshalling, replication and migration. Those mechanisms are left open and can be extended to include user-specified changes and additions.

Middleware. As an approach to building complex distributed systems, middleware has been increasing in popularity. As stated earlier, middleware offers a way to offer appropriate abstractions and services between the application and the operating system/network layers. By adding reflective capabilities to the "middle tier",

reflective middleware [BCRP98] [CBC98] (see picture?), allows the middleware to reconfigure itself so as to be responsive to the changing needs of layers with which it is interacting. Such an approach has been taken in [BCC+99] to support QoS adaptation within a CORBA-based environment. The advantage is that it provides for "agile reconfiguration" of the middleware itself, thus facilitating adaptive support services that can change to suit the reality of its operating environment.

Operating Systems. The application of reflection and object-orientation to distributed operating system design formed the basis for two generations of operating systems,: (1) the first, a research OS (and its corresponding distributed object model) called Muse; and (2) its commercial derivative called Aperios[13].

Muse [YTT89] is an object model which provides a reflective architecture for an object-oriented distributed operating system of the same name. The underlying motivation is that an operating system should provide users with an extensible system using high-level abstractions, appearing as a virtual machine with its own programming environment. In Muse, a meta object accepts requests from objects to perform primitive functions such as allocation of local storage and accessing external devices (the meta object level provides a virtual machine environment, while a meta meta object corresponds to an actual processor). Inter-object communication is therefore location independent while porting Muse requires only modifying any machine-dependent part of the processor's meta meta object.

Aperios [Yok92] [Yok99] is based on the notion that an operating system should use object-orientation in order to encapsulate all shared and/or protected entities. Conversely, they also claim that the system should provide mechanisms to inspect object internals in order to facilitate debugging and object management (such as scheduling). In Aperios, each object has a meta-space which contains meta objects. Every system service is implemented by meta objects; for example, an object's virtual storage is a collection of "(memory) segment" meta objects managed by a virtual segment manager meta object. As these meta objects are also objects, they have meta-spaces as well. The framework is characterized by object/meta object separation, meta-hierarchy and object migration (which is equivalent to transferring metaobjects between meta-spaces).

6.2.4.2 Real-Time Systems

Within the development of Aperios, related research [BMSM95] has been conducted to address real-time concerns and how reflection plays a role within it. Reflection was introduced as a main principle within the Spring real-time operating system [SR89] [SR95]. Used as a fundamental principle and organizational theme for system architecture, the synergy that comes from combining reflection, admission control (for guarantees), and planning (for scheduling) established a new paradigm for real-time systems [Sta99]. In particular, reflection addresses a key real-time systems dilemma: the need for abstraction to deal with complexity versus

---

[13] Readers may be more familiar with one of the other names this operating system has been given over its development: Muse (as the initial research project), Aperos and Apertos (in-term development names) and finally, Aperios (as it is used commercially within Sony™ products) [Yok99].

the need for specific, low-level details associated with real-time requirements [Sta99]. Consequently, the use of reflection within real-time systems tends to parallel that within distributed computing.

6.2.4.3 User-Level Concerns

Reflection has also been used to provide support for user flexibility in terms of the user interface design and the customization of toolkit functionality. Within CSCW, the use of reflection was first proposed by Dourish [Dou92] [Dou93] [Dou95a] [Dou95b] [Dou95c] as a methodology to "re-orient" the design and structure of interactive systems around user needs and participation. The principle focus of his work was the use of reflection as a methodology in the development of CSCW toolkits. Subsequent work by Tolone [Tol96] applied reflective techniques in the development of a collaborative specification system.

User Interfaces. Silica [Rao91] is the CLOS[14]-based portable window system layer of the Common Lisp Interface manager. Silica was the first project to significantly illustrate the notion of implementational reflection towards the realization of open implementation. Silica presents a base-level interface which consists of windows that support input and output operations, while the meta-level interface consists of components which implement base-level behavior and the object-oriented protocols which specify how these components interact. Therefore, Silica allowed the run-time customization of user interface windows and their component widgets.

CSCW Toolkits. The application of reflection to the development of a CSCW toolkit (e.g. user interface widgets, etc.) was motivated in order to support more adaptive and modifiable toolkits than are the traditional norm [Dou96]. By providing generic toolkits that could be adapted in well-defined ways, CSCW applications based on them could be modified on-the-fly to meet their specific needs. Conversely, doing so using traditional techniques required the application to be rebuilt using a different toolkit in a developer-centric rather than user-centric manner.

As an example, some toolkits for CSCW systems have offered different degrees of support for synchronization [Dou94] [DB92]. Again, the traditional approach was to define a toolkit based on a specific synchronization technique (such as locking); this technique was then used throughout any system based on that toolkit. If the provided synchronization model was inappropriate, the application would simply have to use a different toolkit. Similarly, with the integration of different functionality in one toolkit, users were often forced to use whatever collection of techniques the toolkit developers provided. For example, if the toolkit provided a particular UI metaphor with support for a specific database model, the users were required to use them as they were combined within the toolkit itself.

The use of reflection in such a toolkit, however, was intended to allow users to modify (via well-defined interfaces) portions of the toolkit from within the embedding application to provide a tailorable system. For

---

[14] Common Lisp Object System – part of the ANSI Common Lisp standard.

example, the user could re-implement the granularity of a scroll bar widget for a specific application. In particular, the Prospero toolkit [Dou95a] [Dou95b] [Dou95c] [Dou96] deals with applying reflection to issues of data distribution, user interface tailoring and conflict management. The use of reflection in this work is of the implementational variety and offers a means by which users can adjust toolkit functions between applications. A major difficulty with Prospero is its implementation in (and reliance on) the Lisp language to obtain its reflective abilities. Additionally, while Prospero is aimed at building applications, this work provides a reflective framework for managing relationships between entities not just the configuration of a specific application. Therefore, while application-level use is supported, the granularity of reflection within this work is much finer and offers a way to build, maintain and integrate functional as well as behavioural relationships.

Additional differences between Dourish's work and that in this dissertation include the emphasis on:

Multimedia: This work has a stronger interest and focus on multimedia issues, including multimedia synchronization and the potential application of the RASCAL behavioural management approach to mixed mode synchrony. Conversely, Dourish's work is aimed more at a human factors level than that found in this thesis.

Intelligence: Dourish proposes the idea of "accounts" which suggests the ability to provide configuration management based on individual user settings. RASCAL proposes similar capabilities, but bases them on the potential to embed intelligence in the reflective behavioural management tower. As a result, customizable and adaptive behavioural changes for different users and applications based on both user preferences, system resources, or artificial intelligence policy logic offer a more expandable, evolutionary and potentially intelligent approach to this issue.

Collaborative Specification Framework. The Introspect framework by Tolone [Tol96] is meta-level model which allows users to specify "settings for a collaborative context". It is characterized as a meta-level framework because it includes an architecture which enables collaboration support (using a collection of Web applets) and a related specification environment (which specifies how collaborative efforts are organized). The system utilizes implementational reflection to integrate its three levels: the *definition* level causally affects its *specification* level which causally affects the *instantiation* level. Using "templates" of coarse-grain activities, activities and the relationships between them are defined, specified and eventually instantiated in order to achieve a collaborative goal (i.e. realize the collaborative context). An example of Introspect's target "audience" is the collaboration involved in achieving a "New Product Release". To achieve such a goal, activities such as "departmental meetings", "documentation updates", "system upgrades" and "marketing adjustments" would be defined and related to each other in how they result in the release of a new software product. Changes in the how a new product is released would therefore be reflected to these related tasks. This work therefore provides an approach to organizing and "assembling" collaborative tasks. Doing so in a generic and versatile manner, it can be thought of as an "collaborative construction set" in which users specify component activities (including both synchronous and asynchronous) and evolve them over time.

In relation to this dissertation, the Introspect framework does not address any system level issues, nor does it address the integration of multimedia into a collaborative environment. Additionally, it does not address the provision of specific behaviours for the activities it organizes. That is, it provides a coarse level of abstraction that does not deal with low-level issues like that of synchronization. Furthermore, while highly versatile, its specification technique itself is quite complex. Therefore, in general, while being similar in terms of a "compositional theme", the level of abstraction and target audience of Talone's framework is different from that of RASCAL. For example, Talone does not address behavioural specifications in relation to explicit real-time entities.

Therefore, it is important to note that while the use of reflection in CSCW has been proposed before, it has been primarily used as to methodology for toolkit design or collaborative task organization. As well, its use within middleware systems has been applied to the management of QoS issues. It has not been applied to the behavioural management of collaborative multimedia systems (including their synchronization and coordination issues) at the level of detail and permeation as is done within RASCAL. Additionally, while offered as a possible technique to address the user-centric nature of interactive systems, it has not been used as a principle methodology to bridge between system design, implementation and execution. Beyond the use of implementational reflection; this dissertation offered a model by which entities regulate their behaviour meta-level via computational reflection which is causally connected to their execution base-level achieved through implementational reflection. Consequently, these other uses of reflection offer validity to the notion of employing such techniques within a media space coordination architecture and offer a springboard for further study.

6.3 Summary

This chapter has discussed RASCAL by providing a qualitative evaluation of the framework and the MSpace conceptual prototype. Observations about the framework and its potential as a basis for future collaborative media systems was offered.

	Technology	Function	Behaviour	Management	Integration	Deficiency		Interesting Feature
MME	TMM et al.	Augmented reality	Agent-based; dynamic/adaptive	Unclear	UI	Specialized hardware		Emotive computing
	ALIVE		Autonomous intelligent agents	Agent-based	UI/Agent	Single user; specialized hardware		Intelligence/immersive "magic mirror" interface
	TELEPORT		Per participant	Unclear	UI	Specialized hardware		Immersive copresent teleconferencing
CVE	Place Holder	Virtual Reality	Inherent per entity within the environment	Unclear	UI	Socially oriented conceptualization		"marking"-based tracking/navigation system; including voice
	OC/SPLINE					Unicast		Locale partitioning
	MASSIVE2					Specialized hardware		3rd party objects
	DIVE					Scalability of reliable multicast		Dynamically shared databases
	NPSNET					Military oriented		World partitioning
CSCW	JETS	Java applets & applications	Inherent per applet	Session management applet	Functional Container	Java-specific; applet-dependent function; no behavioural management		Applet sharing, applet-based extensibility
	Jasmine			Integrated session management				Transparent event interception
	JAMM			None/unclear		Java-specific; Swing-dependent		
	Habanero	Custom applets, aka "Hablets"		None/unclear		Java-specific; custom browser		Late comer support
	MBONE	Custom tool suite	Inherent per tool	Session management tool	None; user enforced	Disjoint use and management		Multicast
	GroupKit	Custom tools	Inherent per application	Custom	Custom	TCL-specific Lisp/language dependence	Non multimedia	Group orientation
	Prospero	Custom tools	Dynamic and adaptive	Custom	Custom			Reflective
	Introspect	Web applets	Inherent per applet	Custom control environment	Organization	Exceedingly complex		Meta-level task organization framework
RASCAL		Functional objects; implementation specific	Explicit active behavioural specification using patterns/objects; dynamic/adaptive	Active intelligent network service via behavioural representation	Functional & behavioural integration framework	Non-trivial separation of concerns		Reflective middleware for intelligent behavioural management

Table 6.6: Summary of Various "Collaborative Multimedia" Works

7 CHAPTER SEVEN

SUMMARY AND CONCLUSION: RASCAL IN REVIEW

"The real voyage of discovery consists not in seeking new landscapes, but in having new eyes"
Proust (1871 – 1922)

This final chapter concludes the presentation of the RASCAL framework. The research program is briefly summarized and the various contributions are reviewed and related to specific research activities. Future research directions are then outlined followed by some final thoughts.

7.1 Research Overview and Dissertation Summary

This dissertation has presented a novel approach to the realization of collaborative multimedia systems. Through the fusion of computer supported collaborative work and multimedia technologies, the collaborative media space emerges as a dynamic and versatile platform for distributed collaborative interaction. Inherently broad and open-ended, a media space requires its enabling and supporting infrastructure to be appropriately flexible and malleable. Rather than restrict design or implementation, designers and users must be allowed to realize both the requirements and the potential of a given collaborative scenario. Consequently, RASCAL can address the needs of a diverse cross-section of collaborative situations, such as:

- real-time, synchronous distributed environment
- activity/event orientation
- dynamic membership, group and activity definition
- diverse computational and collaborative multimedia requirements
- mixed behavioural requirements and coupling

To meet these needs, the Reflectively Adaptive Synchronous Coordination Architectural framework was proposed. More commonly referred to as RASCAL, the model offers a reflective middleware approach to the management (both structural and behavioural) of real-time synchronous collaborative media spaces. The model offers a framework and methodology for building systems which integrate behavioural management facilities through the use of an "intelligent" policy-driven communication subsystem. The result is a behavioural-oriented, coordination-based collaborative support infrastructure for building shared media spaces which blend presentational, conversational and interactive multimedia.

Based on the reflective and active-object paradigms, specifics of the RASCAL framework were presented along with its application to various collaborative multimedia scenarios. Subsequently, the framework was illustrated

via MSpace, the conceptual prototype implementation undertaken as part of this research. By blending simulation and implementation technologies to realize the various aspects of the RASCAL framework, MSpace provides one example of various approaches that could be taken vis-à-vis a RASCAL-based system.

The potential of RASCAL is further discussed in terms of realistic implementation and performance considerations as well as its long-term prospects as the basis for intelligent, self-adaptive collaborative behavioural management systems.

7.2 Goals, Objectives, Contributions and Achievements

The focus of this research was the realization of a way to design, construct, manage and evolve collaborative media spaces. These media spaces provide for collaboration utilizing any combination of presentational, conversational and interactive multimedia, with no fixed user interface, application toolset, behavioural model or policy. The model offers a framework and methodology for building systems which integrate behavioural management facilities through the use of a policy-driven communication subsystem. Rather than address specific low-level approaches to data transfer, synchronization or the like, this work focused on how to integrate media space elements in a way that allows run-time evaluation, manipulation, control and evolution of the space. The goal was not to design a specific application or system with an intrinsic but hard to qualify behaviour. Rather, the goal was to design a way to build systems out of specific functional and behavioural entities. The intent being that a system can monitor, adapt, modify and evolve itself, both functionally and specifically in terms of its behaviour, in order to meet changing needs over its lifetime.

This work presented a framework and methodology which can be used to build a versatile and enabling infrastructure for user-level collaborative environments. The proposed methodology does not preclude any existing effort but can be viewed as a potential way to organize, relate and integrate different approaches (including user interface technologies, applications and protocols) as required. The architecture's design, its underlying software engineering and the role of the communication infrastructure are therefore primary considerations and are based on a key underlying principle: management and domain functionality must be strictly separated to facilitate a flexible, modifiable and adaptive system.

In meeting these overall goals, the following objectives (as outlined in Chapter One) and how they were met are summarized in Table 7.1. Given the thesis statement, focus and objectives of this research, this work addressed the need to provide for advanced features and complex behaviours within multimedia collaborative environments not by the use of specific low-level approaches but by the correct software engineering and system construction techniques. It takes a responsibility-driven methodology and combines facets of artificial intelligence, active (intelligent) networks, real-time systems, multimedia and collaborative environments. The resulting contributions of this research, initially outlined in Chapter One, are now overviewed in Table 7.2. They are organized in a semi-hierarchical manner which mirrors their provision within the overall research effort.

Objective	How Met
Illustrate a media space as an integrated cooperative system involving the following elements: • *media* • *human participants* • *technological entities*	Definition of a media space was offered and illustrated via a variety of media-based systems, assorted application scenarios and their fit within collaborative and multimedia paradigms.
Specify a coordination-based media space architecture based on the use of a *coordinated communications subsystem* (CCS).	Presented the role of behavioural management (including coordination) and its role in a collaborative media space. Its provision as a service within the network (the CCS) and as offered via the Proxy Chair (part of the CMSS) were explained. The framework and topology to facilitate this organization was based on a meta-level architecture that utilized reflection in order to integrate and unify appropriately separated concerns within the model.
Define the need, role, essence and elements of a *collaborative multimedia support system* (CMSS) relative to the CCS and the given media space architecture.	
Specify a meta-level behavioural management model utilizing the above media space architecture; this includes the role of: • *reflection* • *meta-technologies* • *intelligent network services*	
Provide an integrated and unified approach to the above concerns and issues within a model called *RASCAL* – the *Reflectively Adaptive Synchronous Coordination Architectural framework*	
Explore the integration of presentational, conversational and interactive multimedia within a unified collaborative multimedia framework • This includes addressing the nature and role of synchronization between live and synthetic streams and their combined use within a media space.	The application of the framework to various scenarios, including a variety of media-specific ones was presented. This included the application of the framework to audio/video synchronization as well as mixed mode synchrony between live and synthetic media.
Illustrate the principles of the RASCAL framework using a conceptual prototype	Presented the MSpace conceptual prototype that implements the various facets of RASCAL as outlined in Chapter Five

Table 7.1: Thesis Objective Summary

7.3 Future Work and Research Directions

The work presented in this dissertation has offered a general and flexible approach to collaborative multimedia system development. Consequently, based on the provided research and its illustrated application, numerous avenues for further research exist. This section therefore briefly highlights several of opportunities for future exploration, improvement and implementation.

7.3.1 Implementation Alternatives

One of the most straightforward additions/improvements to this work would be alternative implementations of a collaborative environment like that of MSpace. Interesting possibilities include those based on Java, those based on augmented and virtual reality as well its those offered within a pervasive computing context.

Contribution		How/Where Achieved
Formulation of the basis of a general design methodology for complex real-time collaborative multimedia systems.		Development of the RASCAL framework and its associated methodology (Chapter Three), illustration of its application to various collaborative scenarios (Chapter Four) and potential implementation possibilities (Chapters Five and Six).
Development of a meta-level framework for the construction of collaborative media space environments		Specification and illustration of the RASCAL framework (Chapters Three and Four) and the MSpace prototype (Chapter Five).
Development of a behavioural-based collaborative media space topological model based on an intelligent communications subsystem.		Specification and illustration of the RASCAL communication topology (Chapter Three).
Use of reflection and meta-technologies within real-time collaborative systems:	As a facilitator of "context-mediated behaviours" through the ability to dynamically replace and adapt behaviours.	Introduced as part of the RASCAL framework in Chapter Three, illustrated by example in Chapter Four and via MSpace in Chapter Five; its future potential is detailed under the notion of "INSAIN Behaviours" in Chapter Six.
	As an enabler of a self-investigative and self-regulating approach to adaptive media space synchronization and coordination	Introduced as part of the RASCAL framework in Chapter Three, illustrated by example in Chapter Four and via MSpace in Chapter Five; its future potential is detailed under the notion of "Reflective Synchrony" in Chapter Six.
	As a facilitator of a behavioural agent approach to active collaborative space management and adaptation	Introduced as a general aspect of the RASCAL framework in Chapter Three (vis-à-vis interaction between active objects) and illustrated by example in Chapter Four.
	As an orthogonal and comprehensive design approach to support the coordination and synchronization of media space elements: • Between live and synthetic streams • For both causal and temporal media synchronization • Facilitating the use of different synchronization models and paradigms	Initially presented within the scope of the general RASCAL framework (Chapter Three) and specifically detailed by example in Chapter Four.

Table 7.2: Thesis Contribution Summary

Based on experience with the existing prototype and the popularity of the Java language, the development of a Java-based RASCAL collaborative application would provide (relative to the current OLE/COM prototype) a more light-weight, platform independent and widely available system. The use of dynamically downloadable applets suits the component-oriented nature of RASCAL and the use of Java also offers interesting potential for integrating RASCAL with other collaborative projects pursued within MCRLab. For example, RASCAL could potentially be used to provide a behavioural management system for projects such as JETS or Jasmine (which were detailed earlier in Chapter Six). Since each of these projects is based on the interception/transmission of events, the use of collaborative applets directly maps to the use OLE-enabled applications and the provision of behaviours as applets would facilitate an interesting approach to downloadable and dynamically instantiable behaviours and policies.

Also popular within collaborative systems is the use of augmented and virtual reality (as briefly shown in Chapter Six). While often developed as standalone systems, the provision of user interface issues at a meta level within RASCAL allows the framework to isolate the difficulties and peculiarities of specific HCI technologies. Given the observation that augmented and virtual reality systems are readily influenced by human (user) conduct, RASCAL also has the potential to provide for adaptive behaviours that can meet the changing reality of user behaviours within such a user-centric system. Consequently, the potential for RASCAL to serve as the behavioural backbone for an immersive collaborative system would provide interesting research opportunities.

Finally, in contrast to the "heavy" user interface technologies typical of immersive reality systems, RASCAL could also prove useful in the context of more light-weight pervasive computing devices. As outlined in Chapter Three, the network service approach provided by RASCAL naturally suits a range of client point of presence technologies, including information appliances like set top boxes. Expanding upon this notion along with the framework's user interface independence, RASCAL's real-time event coordination approach could also be applied as an approach to behavioural management amongst distributed light-weight collaborating devices. Examples could include coordinating input amongst multiple video cameras or even more novel devices such as coordinating interactions between appliances such as an electronic coffee cup [Bei99], a coffee maker and a robot which acts as a waiter to deliver coffee which is automatically brewed when the cup is empty. In such a circumstance, the devices used are extremely generic and specific usage policies are not built-in. Consequently, provision of their coordination by the same service which connects them is more appropriate and facilitates easier management of the entities in a wide range of particular usage scenarios.

7.3.2 Behaviour Facilitation and Specialization

As part of its use of meta objects to abstract and isolate entity properties, RASCAL provides behavioural specifications and policies (generically known as behavioural patterns) that regulate the flow of events which constitute entity behaviours. Consequently, the regulation and accuracy of the actual behaviour is affected by the accuracy of the pattern(s) which manage it. Therefore, the ability to manage a behaviour is directly related to the ability to capture the essence of that behaviour in a pattern that is used to regulate it. Hence, by utilizing an environment which allows the development and evaluation of behavioural patterns would be beneficial.

Initially mentioned in Chapter Six, the availability of a more advanced realization of a CSE/CDE would provide a testbed for the creation of such patterns. Variations include: (1) providing a simulation environment for the creation and testing of such patterns, potentially utilizing mockup user interface components to illustrate how the said pattern actually manifests itself; and (2) automatic pattern recognition based on monitoring existing user input. This automatic recognition could then utilize a database to store these commonly occurring behavioural patterns to build a collection of "behavioural profiles" for known users and activities. Alternatively, simulation could provide the ability to test implementation overhead incurred in specific

behavioural situations (such as overhead resulting from behavioural cascades) and the need to address scalability (such as applied to distibution of the Proxy Chair).

7.3.3 Scalability

The scalability of the RASCAL framework is in large part addressed by its component-based, meta-level architecture. However, as mentioned in earlier chapters, concern over the use of a centralized behavioural management system (i.e. the Proxy Chair) can be addressed through the notion of a distributed Proxy Chair. Briefly outlined in Chapter Three, the realization of a distributed Proxy would require additional research to clarify the numerous issues related to an actual implementation. This would include design and implementation specifications, addressing the size of the Proxy cluster itself as well as exploration of the policies and mechanism for collaborative load distribution (which would be akin to process migration/load balancing in operating systems). The necessity and viability of such an approach would need further study based on reasonable performance data (either based on simulation or a more suitable, light-weight and fluid collaborative environment implementation than that offered by the current prototype).

7.3.4 Intelligence

All of the preceding aspects of future work relative to RASCAL would be well-served by one overriding consideration to realize the framework's full potential: intelligence. As outlined within the previous discussion, the general design of the RASCAL framework is designed to allow the introduction of varying levels of computational complexity to deal with different aspects of entity management in whatever way is most suitable. Consequently, the provision of artificial intelligence capabilities (such as neural nets, expert systems or fuzzy logic) within the meta-level policy and behavioural management apparatus would allow the behavioural management backbone to perform more complex analysis of what is occurring throughout the collaborative environment. Examples could range from trend analysis and automatic behavioural modification to the reallocation of resources. The separation of concerns inherent in the framework allow the provisioning and maturation of such facilitates independent of other system aspects. The result is the potential to deal with high-level behavioural abstractions at more of a "collaborative semantic" level rather than in terms of low-level "algorithmic events". While a speculative direction, the potential of a framework to deal orthogonally with everything from manual to potential autonomous collaborative actions would seem worthy of future consideration.

7.4 Final Thoughts

This work has presented a broadly-based approach to the realization of collaborative media spaces. As an integration of multimedia technology with the tenets of computer supported collaborative work, the collaborative media space abstraction forms the basis for a wide range of collaborative media-based systems.

In facilitating the real world collaborative process, a collaborative framework must unify and enable synergy between its components in a natural, utilitarian and conducive manner. It must provide means to specify, utilize, adapt and evolve as the collaborative process does the same. Therefore, there is a need to emphasize the importance of design, its relation to execution and the ability to maintain and evolve the relationship between them. The RASCAL framework addressed these issues by using a combination of techniques ranging from a component-based, meta-level architecture to the use of reflective computations and the potential application of artificial intelligence.

While the organization and application of RASCAL can sometimes seem rather intricate, the purpose of a research framework is not to strictly meet present day needs; rather, it must be able to facilitate system growth and evolution. Therefore, through its inherent ability to change and adapt, the RASCAL framework is an example of addressing diversity and change via simplicity and appropriate engineering, applying the necessary techniques to facilitate the growing complexity of future system design and expectation.

BIBLIOGRAPHY

[Ack94a] Ackermann, P. (1994). *Direct Manipulation of Temporal Structures in a Multimedia Application Framework.* In Proceedings of ACM MM'94. pp. 51 - 58.

[Ack94b] Ackermann, P. (1994). *Design and Implementation of an Object-Oriented Media Composition Framework.* In Proceedings of IEEE International Computer Music Conference ICMC'94.

[AFPS92] Agha, G., Frolund, S., Panwar, R. and Sturman, D. (1992). *A Linguistic Framework for Dynamic Composition of Dependability Protocols.* Proceedings of the Third IFIP Working Conference on Dependable Computing for Critical Applications. International Federation of Information Processing Societies. pp. 345 - 363.

[AEH88] Ahuja, S. R., Ensor, J. R. and Horn, D. N. (1988). *The Rapport Multimedia Conferencing System.* In Proceedings of the ACM Conference on Office Information Systems. Palto Alto, CA. pp. 1 - 8.

[AY96] Akyildiz, I. F. and Yen, W. (1996). *Multimedia Group Synchronization Protocols for Integrated Services Networks.* IEEE Journal on Selected Areas in Communications. Volume 14, Number 1. pp. 162 - 173.

[Ale79] Alexander, C. (1979). *The Timeless Way of Building.* Oxford University Press.

[AIS77] Alexander, C., Ishikawa, S. and Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press.

[ABH+95] Anderson, D. B., Barrus, J. W., Howard, J. H., Rich, C., Shen, C. and Walters, R. C. (1995). *Building Multi-User Interactive Multimedia Environments at MERL.* IEEE Multimedia. Volume 2, Number 4. pp. 77 - 82. Also available as: Technical Report TR-85-17. Mitsubishi Electric Research Laboratories.

[AH91] Anderson, D. P. and Homsy, G. (1991). *A Continuous Media I/O Server and Its Synchronization Mechanism.* IEEE Computer. Volume 24, Number 10. pp. 51 - 57.

[App00] Appleton, B. (2000). *Patterns and Software: Essential Concepts and Terminology.* http://www.enteract.com/~bradapp/docs/patterns-intro.html. An earlier version appeared in Object Magazine Online (Volume 3, Number 5, May 1997).

[BF93] Bandinelli, S. and Fuggetta, A. (1993). *Computational Reflection in Software Process Modeling: The SLANG Approach.* Proceedings of the 15th International Conference on Software Engineering. pp. 144 - 154.

[BWA96] Barrus, J. W., Waters, R. C. and Anderson, D. B. (1996). *Locales: Supporting Large Multi-User Virtual Environments.* IEEE Computer Graphics and Applications, Volume 16, Number 6. pp. 50 – 57.

[BCL+93] Baxter, N., Chien, H., Loreen, A., Marshall, K. and Baraniuk, S. (1993). *OPNet Tutorial Manual.* BDM International Incorporated and MIL 3 Incorporated. Washington, DC.

[BSSS97] Begole, J., Struble, C., Shaffer, C. and Smith, R. (1997). *Transparent Sharing of Java Applets: A Replicated Approach.* In Proceedings of the Symposium on User Interface Software and Technology. ACM Press. pp. 55 - 64.

[Bei99] Gellersen H-W., Beigl, M. and Krull, H. (1999). *The MediaCup: Awareness Technology Embedded in an Everyday Object.* In Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC99).

[BP93] Bekker, C. and Putter, P. (1993). *Reflective Architectures: Requirements for Future Distributed Environments.* Proceedings of the Fourth Workshop on Future Trends of Distributed Computing Systems. pp. 112 - 118.

[Ber96] Berstein, P. A. (1996). *Middleware: A Model for Distributed Services.* Communications of the ACM. Volume 39, Number 2. pp. 86 - 97.

[BG81] Bernstein, P. A. and Goodman, N. (1981). *Concurrency Control in Distributed Database Systems.* In Readings in Database Systems, M. Stonebraker, ed. Morgan Kaufmann: San Francisco, CA. pp. 230 - 248.

[BHG87] Bernstein, P. A., Hadzilacos, V. and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems.* Addison-Wesley: Reading, Massachusetts.

[BCC+92] Blair, G. S., Campbell, A., Coulson, G., Garcia, F., Hutchinson, D., Andrew, S. and Shepard, D. (1992). *Orchestration Services for Continuous Media Communications.* Technical Report MPG-92-54. Distributed Multimedia Research Group, University of Lancaster.

[BCC+99] Blair, G. S., Costa, F., Coulson, G., Delpiano, F., Duran, H., Dumant, B., Horn, F., Parlavantzas, N. and Stefani, J-B. (1999). *The Design of a Resource-Aware Reflective Middleware Architecture.* In Proceedings of Reflection'99: Second International Conference on Meta-Level Architectures and Reflection. Springer-Verlag LNCS 1616. pp. 115 - 134.

[BCRP98] Blair, G. S., Coulson, G., Robin, P. and Papathomas, M. (1998). *An Architecture for Next Generation Middleware.* In Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98). Lake District, UK, Editors: Davies, N., Raymond, K., Seitz, J., Springer-Verlag, 1998.

[BS96] Blakowski, G. and Steinmetz, R. (1996). *A Media Synchronization Survey: Reference Model, Specification and Case Studies.* IEEE Journal on Selected Areas in Communications. Volume 14, Number 1. pp. 5 - 35.

[Bla97] Blaszczak, M. (1997). *Professional MFC with Visual C++ 5.* Wrox Press.

[BHI93] Bly, S. A., Harrison, S. R. and Irwin, S. *Media Spaces: Bringing People Together in a Video, Audio and Computing Environment.* Communications of the ACM. Volume 36, Number 1. pp. 28 – 47.

[BL94] Boucher, J. A. and Little, T. D. C. (1994). *An Adaptive Real-time Software-Only Player for the MPEG-I Video Stream.* Technical Report. Multimedia Communications Lab, Boston University.

[Box97] Box, D. (1997). *The Active Template Library Makes Building Compact COM Objects a Joy.* Microsoft Systems Journal. Volume 12, Number 6. Also available via MSJ/MSDN Online at http://www.microsoft.com/msj.

[Bro95] Brockschmidt, K.. (1995). *Inside OLE, 2nd Edition.* Microsoft Press.

[BK96] Brown, N. and Kindel, C. (1996). *Distributed Component Object Model Protocol -- DCOM/1.0.* Microsoft Press.

[BMA97] Brugali, D., Menga, G. and Aarsten, A. (1997). *The Framework Life Span.* Communications of the ACM. Volume 40, Number 10. pp. 65 – 68.

[BMSM95] Bryce, R. W., Murata, K., Shoja, G. C. and Manning, E. G. (1995). *Porting and Enhancements of a Real-time Object-Oriented Operating System.* In Proceedings of PacRim'95.

[BL91] Bulterman, D. C. A. and van Liere, R. (1991). *Multimedia Synchronization and Unix.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video. (ed. R. G. Herrtwich). pp. 108 - 119. Heidelberg, Germany: Springer-Verlag.

[Bur94] Burkow, T. M. (1994). *Operating System Support for Distributed Multimedia Applications; A Survey of Current Research.* Technical Report (Pegasus Paper 94-8). Faculty of Computer Science, University of Twente.

[BMR+96] Buschman F., Meunier R., Rohnert H., Sommerland P. and Stall M. (1996). *Pattern-Oriented Software Architecture - A System of Patterns.* Wiley and Sons.

[CCGH92] Campbell, A., Coulson, G., Garcia, F. and Hutchinson, D. (1992). *Orchestration Services for Distributed Multimedia Synchronization.* Technical Report MPG-92-52. Distributed Multimedia Research Group, University of Lancaster.

[CKV+99] Campbell, A.T., Kounavis, M.E., Vicente, J., Villela, Miki, K. and H. De Meer, "A Survey of Programmable Networks. ACM SIGCOMM Computer Communication Review. Volume 29, Number 2.

[CF99] Camurri, A. and Ferrentino, P. (1999). *Interactive Environments for Music and Multimedia.* Multimedia Systems. Springer-Verlag. Volume 7, Number 1. pp. 32 - 47.

[CGJ+98] Chabert, A., Grossman, E., Jackson, L., Pietrowicz, S. and Seguin, C. (1998). *Java Object Sharing in Habanero.* Communications of the ACM. Volume 41, Number 6. pp. 69-76.

[Cha96] Chappell, D. (1996). *Understanding ActiveX and OLE.* Microsoft Press.

[CP92] Chen, S. W. and Pu, C. (1992). *A Structural Classification of Integrated Replica Control Mechanisms.* Technical Report CUCS-006-92. Department of Computer Science, Columbia University.

[CHSV97] Codenie, W., De Hondt, K., Steyaert, P. and Vercammen, A. (1997). *From Custom Applications to Domain-Specific Frameworks.* Communications of the ACM. Volume 40, Number 10. pp. 70 – 77.

[CRG97] Communications Research Group.. (1997). Department of Computer Science, University of Nottingham. http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-2

[CBC98] Costa, F., Blair, G. S. and Coulson, G. (1998). *Experiments with Reflective Middleware.* In Proceedings of ECOOP Workshop on Reflective Object-Oriented Programming and Systems (ROOPS'98). Brussels, Springer-Verlag.

[Cou99] Coulson, G. (1999). *A Configurable Multimedia Middleware Platform.* IEEE Multimedia. IEEE Press. Volume 6, Number 1. pp. 62 - 76.

[CFS+94] Craighill, E., Fong, M. W., Skinner, K., Lang, R. E. and Gruenefeldt, K. (1994). *SCOOT: An Object-Oriented Toolkit for Multimedia Collaboration.* In Proceedings of ACM MM'94. pp. 41 - 49.

[CMB+90] T. Crowley, P. Milazzo, E. Baker, H. Forsdick and R. Tomlinson. (1990). *MMConf: An Infrastructure for Building Shared Applications.* Proceedings of CSCW'90. pp. 637 - 650.

[DNNR93] Dannenberg, R. B., Neuendorffer, T., Newcomer, J. M. and Rubine, D. (1993). *Tactus: Toolkit-Level Support for Synchronized Interactive Multimedia.* Multimedia Systems. Volume 1, Number 2. pp 77 - 86.

[DMNS97] Demeyer, S., Meijler, T. D., Nierstrasz, O. and Steyaert, P. (1997). *Design Guidelines for 'Tailorable' Frameworks.* Communications of the ACM. Volume 40, Number 10. pp. 60 – 64.

[DS84] Des Rivieres, J. and Smith, B.C. (1984). *The Implementation of Procedurally Reflective Languages.* Proceedings of the 1984 ACM Symposium on Lisp and Functional Programming. Austin, Texas. August 1984. pp. 348 - 355.

[DG97] Dommel, H. P. and Garcia-Luna-Aceves, J. J. (1997). *Floor Control for Multimedia Conferencing and Collaboration.* Multimedia Systems. Springer-Verlag. Volume 5, Number 1. pp. 23 - 38.

[Dou92] Dourish, P. (1992). *Computational Reflection and CSCW Design.* Technical Report EPC-1992-102. Rank Xerox Research Centre. Cambridge, England.

[Dou93] Dourish, P. (1993). *Designing for Change: Reflective Metalevel Architectures for Deep Customization in CSCW.* Technical Report EPC-1993-102. Rank Xerox Research Centre. Cambridge, England.

[Dou94] Dourish, P. (1994). *A Divergence-Based Model of Synchrony and Distribution in Collaborative Systems.* Technical Report EPC-1994-102. Rank Xerox Research Centre. Cambridge, England.

[Dou95a] Dourish, P. (1995). *Developing A Reflective Model of Collaborative Systems.* ACM Transactions on Computer-Human Interaction. Volume 2. pp. 40 - 63.

[Dou95b] Dourish, P. (1995). *The Parting of the Ways: Divergence, Data Management and Collaborative Work.* Technical Report EPC-1995-105. Rank Xerox Research Centre. Cambridge, England.

[Dou95c] Dourish, P. (1995). *Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaboration Toolkit.* Technical Report EPC-1995-106. Rank Xerox Research Centre. Cambridge, England.

[Dou96] Dourish, P. (1996). *Open Implementation and Flexibility in CSCW Toolkits.* PhD Thesis. Department of Computer Science. University College London. London, England. Also published as Technical Report EPC-1996-106. Rank Xerox Research Centre. Cambridge, England.

[DB92] Dourish, P. and Bellotti, V. (1992). *Awareness and Coordination in Shared Work Spaces.* Technical Report EPC-92-101. Rank Xerox Research Centre. Cambridge, England. Also published in Proceedings of the ACM Conference on Computer-Supported Cooperative Work CSCW'92 (Toronto, Canada).

[EGR91] Ellis, C. A., Gibbs, S. J. and Rein, G. L. (1991). *Groupware: Some Issues and Experiences.* Communications of the ACM. Volume 34, Number 1. pp. 38 - 58.

[EN94] Elmasri, R. and Navathe, S. B. (1994). *Fundamentals of Database Systems.* Second Edition. Benjamin/Cummings Publishing: Redwood City, CA.

[ESGS00] El Saddik, A., Shirmohammadi, S., Georganas, N. D. and Steinmetz, R. (2000). *JASMINE: Java Application Sharing in Multiuser INteractive Environments.* In Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS2000).

[Eri94] Eriksson, H. (1994). *MBone: The Multicast Backbone.* Communications of the ACM. Volume 37, Number 8. pp. 56 - 60.

[FS97] Fayad, M. E. and Schmidt, D. C. (1997). *Object-Oriented Application Frameworks.* Communications of the ACM. Volume 40, Number 10. pp. 32 – 38.

[Fla99] Flanagan, D. (1999). *Java in a Nutshell.* O'Reilly and Associates Incorporated.

[FK97] Fowler, M. and Scott, K. (1997). *UML Distilled.* Addison-Wesley.

[FW84] Friedman, D.P. and Wand, M. (1984). *Reification: Reflection Without Metaphysics.* Proceedings of the 1984 ACM Symposium on Lisp and Functional Programming. pp. 348 - 355.

[Fuj90] Fujimoto, R. M. (1990). *Parallel Discrete Event Simulation.* Communications of the ACM. Volume 33, Number 10. pp. 30 - 53.

[GHJV94] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley.

[GS96] Gawthrop, P. and Smith, L. (1996). *Metamodelling: Bond Graphs and Dynamic Systems.* Prentice-Hall, Inc. Summary available at: http://www.mech.gla.ac.uk/~peterg/books/96/book_summary/node6.html.

[Geo94] Georganas, N. D. (1994). *Synchronization Issues in Multimedia Communications.* Technical Summary. Multimedia Communications Research Laboratory. University of Ottawa.

[Gib91] Gibbs, S. (1991). *Composite Multimedia and Active Objects.* In OOPSLA'91. pp. 97 - 112. Phoenix, Arizona: ACM Press.

[GAB99] Gibbs, S. J., Arapis, C. and Breiteneder, C. J. (1999). *TELEPORT – Towards Immersive Copresence.* Multimedia Systems. Springer-Verlag. Volume 7, Number 3. pp. 214 - 221.

[GDT91] Gibbs, S., Dami, L. and Tsichtritzis, D. (1991). *An Object Oriented Framework for Multimedia Composition and Synchronization.* In Multimedia - Principles, Systems and Applications. (ed. L. Kjelldahl). Springer.

[GA86] Goodman, G. O. and Abel, M. J. (1986). *Collabortion Research in SCL.* In the Proceedings of the ACM First Conference on Computer-Supported Cooperative Work. Austin, TX. pp. 246 - 251.

[GA91] Govindan, R. and Anderson, D. P. (1991). *Scheduling and IPC Mechanisms for Continuous Media.* In Proceedings of the 13th ACM Symposium on Operating System Principles. pp. 68 - 80. California, USA: ACM.

[GM94] Greenberg, S. and Marwood, D. (1994). *Real-Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface.* Research Report 94/534/03. Department of Computer Science, University of Calgary. Calgary, Alberta, Canada.

[GR98] Greenberg, S. and Roseman, M. (1998). *Groupware Toolkits for Synchronous Work.* Computer-Supported Cooperative Work, Trends in Software Series. M. Beaudouin-Lafon, (Ed.) Wiley & Sons Ltd. pp. 135 - 168. Also published as Technical Report 96-589-09.

[GB97] Greenhalgh, C. and Benford, S. (1997). *A Multicast Network Architecture For Large Scale Collaborative Virtual Environments.* In Proceedings of Multimedia Applications, Services and Techniques – ECMAST'97. Springer-Verlag. LNCS 1242. pp. 113 – 128.

[Gri97] Grimes, R. T. (1997). *Professional DCOM Programming.* Wrox Press.

[GMHA95] Grundy, J. C., Mugridge, W. B., Hosking, J. G. and Apperley, M. D. (1995). *Coordinating, Capturing and Presenting Work Contexts in CSCW Systems.* In Proceedings of OZCHI'95. Wollongon, Australia.

[GG95] Gutwin, C. and Greenberg, S. (1995). *Workspace Awareness in Real-Time Distributed Groupware.* Technical Report 95-575-27. Department of Computer Science, University of Calgary. Calgary, Alberta, Canada.

[Hag96] Hagsand, O. (1996). *Interactive Multi-User Virtual Environments in the DIVE System.* IEEE Multimedia. Volume 3,. Number 1. pp. 33 – 39.

[Här84] Härder, T. (1984). *Observations on Optimistic Concurrency Control. Information Systems.* Volume 9, Number 2. pp. 111 - 120.

[HM89] Harrison, S. and Minneman, S. (1989). *The Media Space: An Electronic Setting for Design.* Technical Report SSL-89-63, Xerox Corporation, Palo Alto.

[HM00] Hummes, J. and Merialdo, B. (2000). *Design of Extensible Component-based Groupware.* CSCW – The Journal of Collaborative Computing (Special Issue on Tailoring). Volume 9, Number 1. pp. 53 - 74.

[Ibr91] Ibrahim, M. H. (1991). *Report on the OOPSLA/ECOOP'90 Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming.* In OOPSLA/ECOOP'90 Proceedings Addendum. pp. 73 - 80.

[Ibr92] Ibrahim, M. H. (1992). *Reflection in Object-Oriented Programming.* International Journal on Artificial Intelligence Tools. Volume 1, Number 1. pp. 117 - 136.

[IEE96] IEEE. (1996). IEEE JSAC Special Edition on Multimedia Synchronization. IEEE. Volume 14, Number 1.

[ILM88] Industrial Light and Magic. (1988). http://www.geocities.com/SiliconValley/Bay/2933/favoritestudios.html.

[ITU92] ITU-T. (1992). *Principles of Intelligent Network Architecture.* Recommendation Q.1201. Geneva, Switzerland.

[Jah95] Jahanian, F. (1995). *Run-Time Monitoring of Real-Time Systems.* In Advances in Real-Time Systems, Sang H. Son, ed. Prentice-Hall: Englewood Cliffs, NJ. pp. 435 - 460.

[Jef85] Jefferson, D. R. (1985). *Virtual Time.* ACM Transactions on Programming Languages and Systems. Volume 7, Number 3. pp. 404 - 425.

[Joh97] Johnson, R. E. (1997). *Frameworks = (Components + Patterns).* Communications of the ACM. Volume 40, Number 10. pp. 39 – 42.

[KS99] Karsai, G. and Sztipanovits, J. (1999). *A Model-Based Approach to Self-Adaptive Software.* IEEE Intelligent Systems. Volume 14, Number 3. pp. 46 – 53.

[KP96] Kiczales, G. and Paepcke, A. (1996). *Open Implementations and Metaobject Protocols.* Tutorial Notes. Xerox Corporation.

[KK88] Kramer, K. L. and King, J. L. (1988). *Computer-based Systems for Cooperative Work and Group Decision Making.* ACM Computing Surveys. Volume 20, Number 2. pp. 115- 146.

[KR81] Kung, H. T. and Robinson, J. T. (1981). *On Optimistic Methods for Concurrency Control.* ACM Transactions on Database Systems. Volume 6, Number 2. pp. 213 - 226.

[LHG93] Lamont, L., Henderson, G. and Georganas, N. D. (1993). *A Multimedia Real-Time Conferencing System: Architecture and Implementation.* In Proceedings of IBM CASCON'93. Toronto, Ontario. pp. 64 - 72.

[LLBG96] Lamont, L., Li, L., Brimont, R. and Georganas, N. D. (1996). *Synchronization of Multimedia Data for a Multimedia News-on-Demand Application.* IEEE Journal on Selected Areas in Communications. Volume 14, Number 1. pp. 264 - 278.

[Lam78] Lamport, L. (1978). *Time, Clocks and the Ordering of Events in a Distributed System.* Communications of the ACM. Volume 21, Number 7. July 1978. pp. 558 - 565.

[LST94] Laurel, B., Strickland, R. and Tow, R. (1994). *PlaceHolder: Landscape and Narrative in Virtual Environments.* ACM Computer Graphics. Volume 28, Number 2. pp. 118 - 126.

[LL90] Lauwers, J. C. and Lantz, K. A. (1990). *Collaboration Awareness in Support of Collaboration Technology: Requirements for the Next Generation of Shared Window Systems.* In Proceedings of CHI'90. ACM Press. pp. 303 – 311.

[LBC+92] Leopold, H., Blair, G., Campbell, A., Coulson, G., Dark, P., Garcia, F., Hutchinson, D., Singer, N. and Williams, N. (1992). *Distributed Multimedia Communication System Requirements.* ALCTel-Elin Research.

[LCHS92] Leopold, H., Campbell, A., Hutchinson, D. and Singer, N. (1992). *Towards an Integrated Quality of Service Architecture (QOS-A) for Distributed Multimedia Communications.* ALCTel-Elin Research.

[LA90] Levi, S-T. and Agrawala, A. K. (1990). *Real-Time System Design* . McGraw-Hill: New York, New York.

[LH88] Lewis, B. T. and Hodges, J. D. (1988). *Shared Books: Collaborative Publication Management for an Office Information System.* In Proceedings of the ACM Conference on Office Information Systems. Palto Alto, CA. pp. 197- 204.

[LT91] Leydekkers, P. and Teunissen, B. (1991). *Synchronization of Multimedia Data Streams in Open Distributed Environments.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video. (ed. R. G. Herrtwich). pp. 94 - 104. Heidelberg, Germany: Springer-Verlag.

[LKG93a] Li, L., Karmouch, A. and Georganas, N. D. (1993). *Multimedia Teleorchestra with Independent Sources: Part 1: Temporal Modeling of Collaborative Multimedia Scenarios.* Technical Report 931105. Ottawa-Carleton Institute for Electrical Engineering. University of Ottawa.

[LKG93b] Li, L., Karmouch, A. and Georganas, N. D. (1993). *Multimedia Teleorchestra with Independent Sources: Part 2: Synchronization Algorithms.* Technical Report 931106. Ottawa-Carleton Institute for Electrical Engineering. University of Ottawa.

[LN99a] Li, B. and Nahrstedt, K. (1999). *A Control-based Middleware Framework for Quality of Service Adaptations.* IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms. Volume 17, Number 9. pp. 1632 - 1650.

[LN99b] Li, B. and Nahrstedt, K. (1999). *Dynamic Reconfiguration for Complex Multimedia Applications.* In Proceedings of IEEE International Conference on Multimedia Computing and Systems (IEEE Multimedia Systems'99). pp. 165 - 170.

[LG90] Little, T. D. C. and Ghafoor, A. (1990). *Synchronization and Storage Models for Multimedia Objects.* IEEE Journal on Selected Areas in Communication. Volume 8, Number 3. pp. 413 - 427.

[LG91] Little, T. D. C. and Ghafoor, A. (1991). *Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks.* IEEE Computer. Volume 24, Number 10. pp. 42 - 50.

[LK92] Little, T. D. C. and Kao, F. (1992). *An Intermedia Skew Control System for Multimedia Data Presentation.* In Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video. (ed. P. V. Rangan). pp. 130 - 141. La Jolla, California: Springer-Verlag.

[MZP+94] Macedonia, M., Zyda, M., Pratt, M., Barham, P. and Zeswitz, S. (1994). *NPSNET: A Network Software Architecture for Large Scale Virtual Environments.* Presence. Volume 3, Number 4. pp. 265 – 287.

[Mac91] Mackay, W. E. (1991). *Beyond Iterative Design: User Innovation in Co-Adaptive Systems.* Technical Report EPC-1991-130. Rank Xerox Research Centre. Cambridge, England.

[Mae87] Maes, P. (1987). *Computational Reflection.* PhD Thesis. Laboratory for Artificial Intelligence. Vrije Universiteit Brussel. Brussels, Belgium.

[MBD+95] Maes, P., Blumberg, B., Darrel, T., Pentland, A. and Wexelblat, A. (1995). *Modeling Interactive Agents in ALIVE.* In Proceedings of the Interface Joint Conference On Artificial Intelligence IJCAI-95. pp. 2073 - 2074.

[MC94] Malone, T. W. and Crowston, K. (1994). *The Interdisciplinary Study of Coordination.* ACM Computing Surveys. Volume 16, Number 1. pp. 87 - 119.

[MFK+95] Mandeville, J., Furness, T., Kawahata, M., Campbell, D., Danset, P., Dahl, A., Dauner, J., Davidson, J., Kandie, K. and Schwartz, P. (1995). *GreenSpace: Creating a Distributed Virtual Environment for Global Applications.* In Proceedings of the 2nd IEEE Workshop on Networked Realities. Boston, MA. pp. 53-67.

[McA93] McAffer, J. (1993). *The CodA MOP.* In Proceedings of OOPSLA'93 Reflection Workshop.

[McA95] McAffer, J. (1995). *A Meta-Level Architecture for Prototyping Object Systems.* PhD Thesis. University of Tokyo. Tokyo, Japan.

[MBK+99] McCanne, S., Brewer, E., Katx, R., Rowe, L., Amir, E., Chawathe, Y., Coopersmith, A., Mayer-Patel, K., Raman, S., Schuett, A., Simpson, D., Swan, A., Tung, T-L., Wu, D. and Smith, B. (1999). *Toward a Common Infrastructure for Multimedia-Networking Middleware.* In Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97).

[ME95] Meyer-Boudnik, T. and Effelsberg, W. (1995). *MHEG Explained.* IEEE Multimedia. Volume 2, Number 1. pp. 26 - 38.

[MES93] Meyer, T., Effelsberg, W. and Steinmetz, R. (1993). *A Taxonomy on Multimedia Synchronization.* In Proceedings of the 4th IEEE International Workshop on Future Trends in Distributed Computing Systems. pp. 97 - 103. Lisbon, Portugal: IEEE.

[COM95] Microsoft Corporation. (1995). *The Component Object Model Specification.* Microsoft Press.

[MSC00] Microsoft Corporation. (2000). http://www.microsoft.com/directx.

[MER00] Mitsubishi Electric Research Laboratories. (2000). http://www.merl.com/threads/social

[NS95] Nahrstedt, K. and Smith, J. M. (1995). *The QOS Broker.* IEEE Multimedia. Volume 2, Number 1. pp. 53 - 67.

[NC93] Nigay, L. and Coutaz, J. (1993). *A Design Space for Multimodal Systems – Concurrent Processing and Data Fusion.* In Proceedings of the INTERCHI-93 Conference on Human Factors in Computing Systems. pp. 172 - 178.

[Obj97] ObjecTime (1997). *ObjecTime Developer C++ Guide.* ObjecTime Limited.

[OMS+00] Oliveira, J. C., Mosseini, M., Shirmohammadi, S., Cordea, M., Petriu, E., Petriu. D and Georganas, N. D. (2000). *VIRTUAL THEATER for Industrial Training: A Collaborative Virtual Environment.* In Proceedings of the 4th World Multi-Conference on Circuits, Systems, Communications & Computers (CSCC 2000).

[Ore96] Oreizy, P. (1996). *Issues in the Runtime Modification of Software Architectures.* Technical Report UCI-ICS-TR-96-35. Department of Information and Computer Science, University of California. Irvine, CA.

[OGT+99] Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S. and Wolf, A. L. (1999). *An Architecture-Based Approach to Self-Adaptive Software.* IEEE Intelligent Systems. Volume 14,. Number 3. pp. 54 – 62.

[OO97] Ouellette, M. and Orozco-Barbosz, L. (1997). *Performance Evaluation of a Multiprocessor SCP for Intelligent Network Services.* In Proceedings of IEEE Canadian Conference On Electrical and Computer Engineering. pp. 804 - 807.

[PLV97] Posnak, E. J., Lavender, R. G. and Vin, H. M. (1997). *An Adaptive Framework for Developing Multimedia Software Components.* Communications of the ACM. Volume 40, Number 10. pp. 43 – 47.

[RR92] Ramanathan, S. and Rangan, P. V. (1992). *Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems.* The Computer Journal - Special Issue on Distributed Multimedia Systems. December 1992.

[Rao91] Rao, R. (1991). *Implementational Reflection in Silica.* Proceedings of ECOOP'91. In Lecture Notes in Computer Science #512. Springer-Verlag. pp. 251 - 266.

[RBD+97] Reinefeld, A., Baraglia, R., Decker, T., Gehring, J., Laforenza, D., Ramme, F., Römke, T. and Simon, J. (1997). *The MOL Project: An Open, Extensible Metacomputer.* In Proceedings of the Sixth Heterogeneous Computing Workshop. Geneva, Switzerland. pp. 17- 31.

[Rob95] Robbins, R. W. (1995). *A Model for Multimedia Orchestration.* M.Sc. Thesis. Department of Computer Science. University Of Calgary.

[RG97] Robbins, W. and Georganas, N. D. (1997). *Shared Media Space Coordination: Mixed Mode Synchrony in Collaborative Multimedia Environments.* In Proceedings of the IEEE Multimedia Systems'97. Ottawa, Canada. pp. 466 - 473.

[RG99] Robbins, W. and Georganas, N. D. (1999). *Reflective Media Space Management Using RASCAL.* In Proceedings of Reflection'99: Second International Conference on Meta-Level Architectures and Reflection. Springer-Verlag LNCS 1616. pp. 95 - 114.

[Roo88] Root, R. W. (1988). *Design of a Multi-Media for Social Browsing.* In Proceedings of the Second Conference on Computer-Supported Cooperative Work. Portland, OR. pp. 25 - 38.

[Sch97] Schmid, H. A. (1997). *Systematic Framework Design by Generalization.* Communications of the ACM. Volume 40, Number 10. pp. 48 – 51.

[SKD96] Schnepf, J., Konstan, J. A. and Du, D. H. C. (1996). *Doing FLIPS: FLexible Interactive Presentation Synchronization.* IEEE Journal on Selected Areas in Communications. Volume 14, Number 1. pp. 114 - 125.

[Sch95] Schreiber, R. (1995). *Middleware Demystified.* Datamation. Volume 41, Number 6. pp. 41 - 45.

[SCFJ96] Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V. (1996). *RTP: A Transport Protocol for Real-Time Applications,.* Request for Comments rfc1889.

[SCJ87] Scigliano, J. A., Centini, B. A. and Joslyn, D. L. (1987). *A Real-Time Unix-based Electronic Classroom.* In Proceedings of the IEEE Soueastcon'87. Tampa, FL.

[Sel93] Selic, B. (1993). *An Efficient Object-Oriented Variation of the Statecharts Formalism for Distributed Real-Time Systems.* In Proceedings of CHDL'93: IFIP Conference on Hardware Description Languages and Their Applications. Ottawa, Canada.

[Sel94] Selic, B. (1994). *Prototyping with the ObjecTime Toolset: Theory and Practice.* Presentation Slides. Carleton University.

[SGME92] Selic, B., Gullekson, G., McGee, J. and Engelberg, I. (1992). *ROOM: An Object-Oriented Methodology for Developing Real-Time Systems.* In Proceedings of CASE'92: Fifth International Workshop on Computer-Aided Software Engineering. Montreal, Canada.

[STD94] Selic, B., Tigg, R. and Daoust, D. (1994). *Resolved: High-Level System Models Can be Formally Transformed into Complete and Efficient Real-Time Implementations.* In Proceedings of OOPSLA'94 Workshop on Object-Oriented Real-Time Systems Analysis and Design Issues. Portland, USA.

[SG00] Shirmohammadi, S. and Georganas, N. D. (2000). *Collaborating in 3D Virtual Environments: A Synchronous Architecture.* In Proceedings of the IEEE 9th International. Workshops on Enabling Technological Infrastructure for Collaborative Enterprises (WETICE) Knowledge Media Networking Workshop.

[SOG98] Shirmohammadi, S., Oliveira, J. C. and Georganas, N. D. (1998). *Applet-Based Telecollaboration: A Network-Centric Approach.* IEEE Multimedia. Volume 5, Number 2. pp. 64 - 73.

[Sin89] Singh, B. (1989). Invited Talk to Coordination Systems. The Organizational Computing Conference. Austin, TX.

[SS94] Singhal, M. and Shivaratri, N. G. (1994). *Advanced Concepts in Operating Systems: Distributed, Database and Multiprocessor Operating Systems.* McGraw-Hill Incorporated.

[Smi84] Smith, B.C. (1984). *Reflection and Semantics in Lisp.* In Conference Record of the ACM Symposium on Principles of Programming Languages. ACM Press. pp. 23 - 25.

[Smi85] Smith, B.C. (1985). *Prologue to Reflection and Semantic in a Procedural Language.* In Readings in Knowledge Representation. Morgan Kaufmann Publishers. Los Altos, CA.

[Son00a] Sony Electronics Inc. (2000). http://www.aibo.com.

[Son00b] Sony Electronics Inc. (2000). Personal Communication.

[Sta99] Stankovic, J. A. (1999). *Reflection in Real-Time Systems.* Invited Talk. In Proceedings of Reflection'99: Second International Conference on Meta-Level Architectures and Reflection. Springer-Verlag LNCS 1616. p. 1.

[SR89] Stankovic, J. A. and Ramamritham, K. (1989). *The Spring Kernel: A New Paradigm for Real-Time Operating Systems.* ACM Operating Systems Review. ACM Press. Volume 23, Number 3. pp. 54 - 71. Also available as: *The Spring Kernel: A New Paradigm for Real-Time Systems.* ' IEEE Software. Volume 8, Number 3. pp. 62 – 72.

[SR95] Stankovic, J. A, and Ramamritham, K. (1995). *A Reflective Architecture for Real-Time Operating Systems.* In Advances in Real-Time Systems. (ed. Sang, G. Son). pp. 23 - 38. Prentice-Hall. Englewood Cliffs, NJ.

[Ste99] Stefani, J-B. (1999). *On the Reflective Structure of Information Networks.* Invited Talk. In Proceedings of Reflection'99: Second International Conference on Meta-Level Architectures and Reflection. Springer-Verlag LNCS 1616. pp. 93 - 94.

[Ste90] Steinmetz, R. (1990). *Synchronization Properties in Multimedia Systems.* IEEE Journal on Selected Areas in Communications. Volume 8, Number 3. pp. 401 - 412.

[Ste95] Steinmetz, R. (1995). *Analyzing the Multimedia Operating System.* IEEE Multimedia. Volume 2, Number 1. pp. 68 - 84.

[Ste96] Steinmetz, R. (1996). *Human Perception of Jitter and Media Synchronization.* IEEE Journal on Selected Areas in Communications. Volume 14, Number 1. pp. 61 - 72.

[SE93] Steinmetz, R. and Engler, C. (1993). *Human Perception of Media Synchronization.* Technical Report 43.9310. IBM European Networking Center, Heidelberg, Germany.

[SN95] Steinmetz, R. and Nahrstedt, K. (1995). *Multimedia: Computing, Communications and Applications.* Prentice-Hall Incorporated.

[Stu86] Stults, R. (1986). *Media Space.* Technical Report. Xerox Corporation, Palo Alto.

[Sun00] Sun Microsystems. (2000). http://www.javasoft.com.

[Tan92] Tanenbaum, A. S. (1992). *Modern Operating Systems.* Prentice-Hall Incorporated.

[Tem97] Templeman, J. (1997). *Beginning MFC COM Programming.* Wrox Press.

[Tol96] Tolone, W. J. (1996). *Introspect: A Meta-Level Specification Framework for Dynamic, Evolvable Collaboration Support.* Ph.D. Thesis. Department of Computer Science. University Of Illinois at Urbana-Champaign.

[UMAI94] University of Michigan Artificial Intelligence Laboratory. (1994). *A Survey of Cognitive and Agent Architectures.* http://ai.eecs.umich.edu/cogarch0/common/prop/glassbox.html.

[Vli93] Van Vliet, H. (1993). *Software Engineering: Principles and Practice.* John Wiley.

[Vin97] Vinoski, S. (1997). *CORBA: Integrating Diverse Applications Within Distributed Hetergeneous Environments.* IEEE Communications Magazine. Volume 35, Number 2. pp. 46 - 55.

[Voc97] VocalTech InternetPhone. (1997). http://www.vocaltech.com/products/iphone5/index.html. Commerical Software Product. VocalTech Incorporated.

[WSM+90] Watabe, K., Sakata, S., Maeno, K., Fukuoka, H. and Maebara, K. (1990). *A Distributed Multiparty Desktop Conferencing System and its Architecture.* In Proceedings of the IEEE Phoenix Conference on Computers and Communications. Phoenix, AZ. pp. 386 - 393.

[WAB+96] Waters, R. C., Anderson, D. B., Barrus, J. W., Brogan, D. C., Casey, M. A., McKeown, S. G., Nitta, T., Sterns, I. B. and Yerazunis, W. S. (1996). *Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability.* Technical Report TR96-02a. Mitsubishi Electric Research Laboratories.

[Web89a] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 259.

[Web89b] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 1197.

[Web89c] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 745.

[Web89d] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 141.

[Web89e] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 989.

[Web89f] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 288.

[Web89g] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 434.

[Web89h] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 433.

[Web89i] Webster. (1989). *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster Inc. p. 1049.

[WQG94] Woo, M., Qazi, N. U. and Ghafoor, A. (1994). *A Synchronization Framework for Communication of Pre-orchestrated Multimedia Information.* IEEE Network. Volume 8, Number 1. pp.52 - 61.

[Yav92] Yavatkar, R. (1992). *MCP: A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications.* In Proceedings of the 12th International Conference on Distributed Computing Systems. IEEE Computer Society. pp. 606 – 613.

[YD96] Yang, Z. and Duddy, K. (1996). *CORBA: A Platform for Distributed Object Computing.* Operating Systems Review. Volume 30, Number 2. pp. 4 - 31.

[Yok92] Yokote, Y. (1992). *The Apertos Reflective Operating System: The Concept and its Implementation.* Proceedings of OOPSLA'92. pp. 414 - 434.

[Yok99] Yokote, Y. (1999). *Past, Present and Future of Aperios.* Invited Talk. In Proceedings of Reflection'99: Second International Conference on Meta-Level Architectures and Reflection. Springer-Verlag LNCS 1616. p. 153.

[YTT89] Yokote, Y., Teraoka, F. and Tokoro, M. (1989). *A Reflective Architecture for an Object-Oriented Distributed Operating System.* Proceedings of ECOOP'89. In Lecture Notes in Computer Science #707. pp. 89 - 106.

[ZG99] Zhao, H. and Georganas, N. D. (1999). *An Approach for Stream Transmission over HLA-RTI in Distributed Virtual Environments.* In Proceedings of the IEEE/ACM Third International Workshop on Distributed Interactive Simulation and Real Time Applications (DIS-RT'99). . pp. 67 - 74.

APPENDIX A

MULTIMEDIA ORGANIZATION AND SYNCHRONIZATION NOMENCLATURE

This appendix overviews the organizational and synchronization terminology used within multimedia systems. It is presented as an appendix due to its definitional nature and supportive role to the main text throughout the dissertation.

The variety of media types that constitute multimedia yields a broad diversity in the management, presentation and exchange of information. Since multimedia systems are expected to support a range of different media, there also exists a wide range of processing characteristics that must be handled. These characteristics are often unique to a particular medium and dictate certain kinds of processing environments. As a result, convoluted performance requirements are dictated not only by a given medium's complexity but also the difficulty associated with combining them; it is the combination of different kinds of media within a single environment that dramatically increases the complexity of multimedia systems. For example, audio is very susceptible to delays in data delivery and requires continuity in data availability to avoid noisy reproduction. In contrast, however, video is more often characterized by voluminous data sizes and large computational overheads; data loss may be less traumatic and continuity shifts are less noticeable. Therefore, a multimedia system must allow and compensate for the differences between media so that each can be rendered appropriately.

Because of the wide range of characteristics associated with different data types, media are generally grouped into categories or *classes*. Common classes include image, audio, video, animation and text. Each *media class* yields a specific kind of representation and typically entails certain (range of) behavioural characteristics. Each instance of a medium within a media class is typically identified as a *media object*. As such, it specifies its own unique attributes and encoding scheme. For example, one media object may be an GIF image while another may be encoded as a JPEG, each with its own specific width, height, colour table and bitmap. Within a media object, any renderable element of that object is called a *media unit*[15]; that is, a media object is made up of one or more media units that when rendered (according to any required temporal constraints) constitute the media object. For example, a video clip object would be formed from a series of video frames or individually rendered images. Each frame (or image) would therefore constitute a single media unit.

Within each media class, an actual medium could be represented using any of a variety of encoding schemes. An enormous variety of such schemes exists and each has its own unique processing environment in terms of computational overhead, data size and so forth. Some schemes are quite simple while others are extremely complex. Additionally, there is no guarantee that any set of encoding schemes will work well together (in terms of having similar processing requirements) or that any particular scheme will be well-suited to the

---

[15] *Media units* are also commonly referred to as *Logical Data Units* (or LDUs) in the literature.

software/hardware used to render it. Consequently, the characteristics of a specific media class as well as an individual medium can have a significant impact on their integration into a common system.

For example, pre-recorded *synthetic* media vs. real-time *live* media (as explained in Chapter Two) have computational requirements which mirror their different usage characteristics. However, even within each of these categories, different media and their associated codecs do not have the same computational or behavioural profiles. For example, while the representation of image, audio and text classes is obvious, the distinction between video and animation is less pronounced. In particular, the difference lies not in the motion component of the media but in whether or not it was computer-generated (as in the case of animation) or captured from "the real world" by some sort of digitization process (as in the case of video). Furthermore, not all video sequences can be classified in a similar manner due to the variation in the details of their codecs.

Beyond the classification as either synthetic or live, a medium can be further categorized based on its relationship to time. For example, media such as audio and video must meet strict temporal requirements in order to convey their intended meaning. Conversely, others such as image and text impose no specific temporal requirements on their display. These two super-classes based on a medium's *temporal dimension*, referred to as *time-dependent* and *time-independent* media, play an important role in the behaviour and architecture of a multimedia system. A medium's temporal dimension [BCC+92] [CCGH92] indicates whether or not a medium has an *implicit temporal component* as part of its *cognitive content*. The cognitive content of a medium refers to the meaning derived by the person(s) viewing it. A medium is said to have an implicit temporal component if that medium's interpretation (i.e. cognitive content) is subject to temporal constraints during playback. For example, a still image does not have an implicit temporal component since its meaning is not affected by temporal variations during playback. An audio clip, however, does have an implicit temporal component since variations in playback speed and continuity affect comprehension. Those media with a temporal component are called *continuous media* (as their playback requires continuity to be understood). Media without such a temporal component are called *static* [LG90] or *discrete* [CCGH92]. Because continuous media also tend to be utilized at constant, fixed rates (e.g. a video frame every ≈1/30 second), they have a periodic or "rhythmic" quality; as a result, they are said to be *isochronous* [LCHS92].

Representing a layer of abstraction below objects [MES93], media objects are sometimes discussed in terms of *streams* within the context of synchronization (particularly in the case of continuous media). This is due to their data typically being provided as a "stream of LDUs" in a manner akin to the familiar notion of input/output streams in file handling.

As a result, media synchronization is typically classified in relation to the number of media objects (streams) within a temporal relationship as well as their temporal dimension [BS96]:

- intra-stream vs. inter-stream
- event vs. continuous

Intra-stream synchronization (or *continuity*) refers to synchronization internal to a media stream; i.e. exercising the flow of data within a single stream so that it is played back correctly. Conversely, *inter-stream synchronization* refers to the synchronization between independent media streams; i.e. aligning the playback of two or more streams. These categories refer to the autonomy of synchronization, be that within a single stream or between multiple ones.

Event synchronization denotes the alignment of a media object's start to a specific time or event; for example, the display of an image when the background music starts. *Continuous synchronization*, however, refers to the fine-grain synchrony required by continuous media within the duration of an individual object. For a continuous medium, continuous synchronization is the means by which intra-stream synchronization (continuity) is achieved. On the other hand, the classic "lip sync" problem between audio and video streams is an illustration of continuous inter-stream synchronization. Of course, for continuous inter-stream synchronization to be effective, it is assumed that each stream has also been intra-stream synchronized.

APPENDIX B

REFLECTIVE NOMENCLATURE

This appendix includes an overview of the terms commonly used when discussing reflection and reflective systems. This material is borrowed mostly from [Ibr92] unless otherwise noted and is placed in this appendix since it is of a definition-intensive nature which does not fit the explanatory mode of Chapter 2.

Reflection. The original notion of *reflection* was defined (in a somewhat convoluted manner) as a process within the reflection hypothesis [Smi85]:

> "In as much as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of that world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representations of its own operations and structures."

Subsequently, the essence of reflection was more succinctly captured by Maes within her study of object-oriented reflective systems [Mae87] as being:

> "... the process of reasoning about and/or acting upon oneself".

And ultimately, Smith refined his definition within [Ibr91] in a simpler and more coherent manner:

> "Reflection is an entity's integral ability to represent, operate on and otherwise deal with itself in the same way that it represents, operates on and deals with its primary subject matter."

Base and Meta Levels. A reflective system and the concept of reflection itself are based on computation occurring at a minimum of two levels: (1) the *base* level; and (2) the *meta* level. The *base level* is the problem domain of the system itself, while the *meta level* is a level of abstraction above the original problem domain such that it deals with the base level computation as its subject matter. A meta level computation involves the system inspecting and reasoning about the base-level computation from a higher (meta) level.

Meta System. A *meta system* is a computational system[16] that reasons about and acts upon another computational system.

Object System. The domain of a meta system is called its *object system*, which is represented as data within the meta system.

---

[16] "A computational system is something that reasons about and acts upon some part of the world, called the domain of the system. A computational system represents its domain under the form of data. Its program prescribes how these data should be manipulated" [Mae87].

Reflective System. Intuitively, a reflective system is one which utilizes reflection. Within [Mae87], it was defined within an object-oriented context as "*... a causally connected meta system that has as object system itself*".

Reflective Act. See *Reflective Computation.*

Reflective Computation. A *reflective computation* is one performed by a system when reasoning about or acting upon itself.

Causal Link. Reflective computation allows not only the inspection of base-level computation but also the ability to change it. The results of performing this meta-level computation must therefore be incorporated into the base-level when the system resumes functioning at its base-level. This affectual relationship between the base and meta-levels has been called a *causal link.* Therefore, the causal link[17] is a necessary condition for reflection since it allows reflective (meta) computation to influence and alter application (base) computation. It hilights the need to carefully consider which aspects of a system's domain should be represented as data within its computational system.

Causally Connected System. A *causally connected system* is a system which has its causal (connection) link set once by the implementer. Doing so guarantees that changes which result from the interaction between the computational system and its domain are incorporated automatically without any human interaction. A causally connected system is defined as [Mae87]:

1. *when data of the system change, the entities in the domain represented by these data are affected;* and
2. *when entities in the domain change, the data representing these (or aspects of these) in the system are affected.*

Self-Representation. The *self-representation* of a system is the data that is the causally connected representation of itself. That is, self-representation is the way a system symbolizes/characterizes itself internally. In colloquial terms, it is how the program "sees itself".

Introspection. *Introspection* is the self-examination of the system by using meta-level computations which examine the system at corresponding base levels.

Reification. *Reification* [FW84] refers to the process of representing and passing the contents of a system's base level to its meta level so it can be manipulated as data. If a system is able to *reify*, this means that its executing programs can access and inspect their structure and computing environment. However, there is no guarantee that changes in the inspected items will be incorporated into the executing application program. Consequently, reification is a necessary but insufficient condition for reflection.

---

[17] Referred to as a *Causal Connection Link* by [Mae87].

Intercession. *Intercession* is the act of interceding or intervening (between levels) so as to affect a change between them (typically the modification of meta level data).

Deification. *Deification* is the process by which the system returns to the computation that was being accomplished before a reflective meta computation started. Changes made during the meta computation (i.e. *intercession*) are incorporated back into the base level. Consequently, when combined with reification, the causal link necessary for reflection is achieved.

Reversion. See *Deification.*

Reflective Tower. A *reflective tower* is the group (hierarchy) of meta levels that exist in a system. The tower exists since each meta-level can have its own meta level above it. For example, meta level "M" can have meta level "MM" above it. Therefore, meta level "M" can be considered a 'base level' to meta level "MM" which reifies the computations performed in meta-level "M".

Degree of Introspection. *The degree of introspection* [DS84] is defined as the number of meta levels after which higher levels offer no practical value. That is, the degree of introspection denotes how many levels in a reflective tower "make sense" for a given circumstance.

Computational Reflection. *Computational reflection* is a variant of reflection as applied to general computational systems. Its focus is application of reflective principles to a system's behaviour (i.e. enabling it to change the way it computes). Historically known as *Procedural Reflection* (archaic).

Implementational Reflection. *Implementational reflection* is the application of reflective principles to the construction of software systems in terms of integrating and managing their component modules (through the use of well-defined auxiliary interfaces). See *Open Implementation.*

Open Implementation. *Open implementation* [KP96] is a software design technique in which software components are written to be both reusable and efficient for a wide range of clients. Such modules allow their clients individual control over their own implementation strategy, in effect allowing the client to tailor the module's implementation strategy to better suit their particular needs.