Services, Frameworks, And Paradigms for Distributed Multimedia Applications

Max Mühlhäuser & Jan Gecsei
Universities of Linz and Montréal

A B S T R A C T : The development of distributed multimedia applications is supported by an increasing number of services. While such services pave the way towards sophisticated multimedia support even in distributed systems, using them still makes the task of developers quite tedious. This is because several inconsistent services have to be interfaced in order to reflect different aspects. As a way to alleviate this problem, we make the case for an encompassing framework in which all services would be offered under a unifying paradigm. First we give an overview of existing multimedia services with a focus on distribution, extracting the requirements imposed on multimedia extensions to general frameworks as a set of so-called abstractions. Known development environments for distributed applications are obvious candidates for such encompassing frameworks. We review these based on four popular paradigms: client-server / remote procedure call, object-orientation, hypermedia, and open documents; we also investigate possible multimedia extensions, and discuss the 'expressive power' of the paradigms. In conclusion we propose steps towards an encompassing framework based on a hybrid object/hypermedia paradigm.

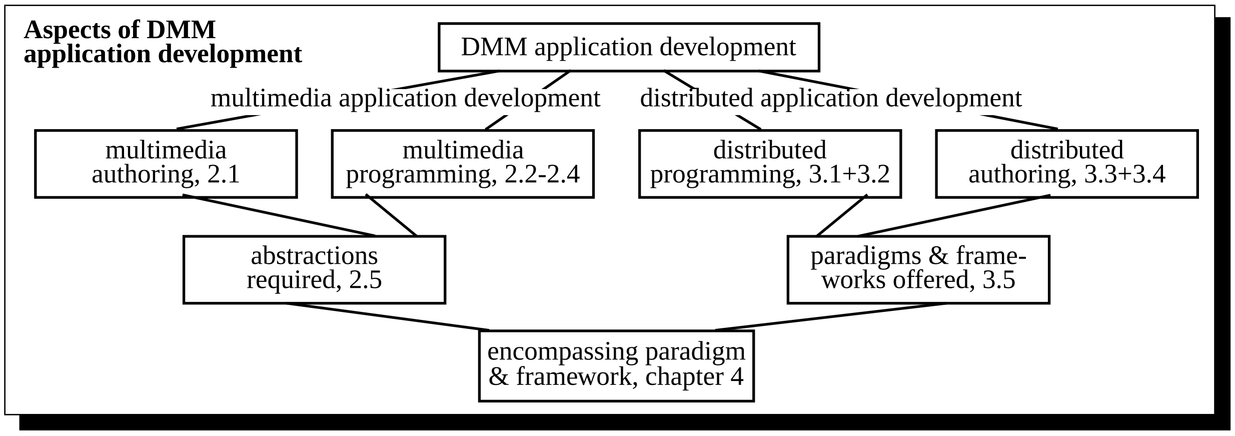
1 Introduction

The multimedia community agrees almost unanimously that both the largest potential and the most serious problems concern the development of distributed multimedia (DMM) applications [13]. This article reviews and compares approaches to the development of such DMM applications. It turns out that this evolving domain inherits from different and rather incompatible domains. On one hand, multimedia applications may either consist mainly of software, requiring a kind of programming activity for their development, or mainly of multimedia data, requiring a kind of authoring activity. On the other hand, multimedia application development has of course first tried to solve problems for stand-alone computers before 'going distributed'; at the same time, the state of the art in distributed application development has greatly advanced, so that the two rather incompatible fields of *multimedia* application development and *distributed* application development have evolved. Just as multimedia application development has progressed in two quite separate flavors, programming and authoring, so has distributed application development: there is the traditional field of *distributed programming* and the handling of distributed multiple-media documents (such as the World-Wide Web). We use the term *distributed authoring* for the latter. In summary, DMM application development has to harmonize four fields: multimedia programming, multimedia authoring, distributed programming, and distributed authoring.

Since we want to discuss ways of harmonizing these fields, we omit from our review details about the myriad of self-contained authoring tools like Macromedia Director™ that exist on multimedia PCs, making section 2.1 rather short; instead, we distill from the multimedia authoring and programming support the key abstractions (see below) required for adequate development support. As a look at the state of the art in distributed application development teaches, we must aim at an encompassing framework that comprises tools and services and for which the design center is a common paradigm such as 'distributed object-oriented programming'. Sidebar 'aspects of DMM application development' shows the four fields that intervene in DMM development and how they are treated and integrated in this article.

Four basic terms must be clarified as used in the article. The term *service* here is to be taken in its broadest sense, covering facets that may be called function or class library, toolkit, and even framework. A DMM service, then, is a set of functions offered to developers which covers DMM-specific aspects. *Abstractions* shall hereafter denote intuitive, self-contained descriptions of functional units within a service, often reflected in software as a set of abstract data types and operations. As multimedia services are still in their infancy, they hardly make abstractions very explicit; rather, most of them offer a single common abstraction to the programmer: the application programming interface, API. An *encompass-*

(ii) a consistent, simple, and homogeneous view onto these services, following a certain mental model; the latter is called *paradigm* here. While APIs represent abstractions which are neither encompassing several services nor based on a precise mental model, encompassing frameworks do. Brief, paradigms below denote a way to describe virtually all components of frameworks and applications, abstractions denote a way to describe a specific functionality.


Aspects of DMM application development
DMM application development
multimedia application development
distributed application development
multimedia authoring, 2.1
multimedia programming, 2.2-2.4
distributed programming, 3.1+3.2
distributed authoring, 3.3+3.4
abstractions required, 2.5
paradigms & frameworks offered, 3.5
encompassing paradigm & framework, chapter 4

2 Multimedia Application Development

In this chapter we briefly review services available to developers of multimedia applications. As discussed in the introduction, we distinguish multimedia authoring and programming, distilling general abstractions used. As to multimedia programming, we investigate three classes of support systems particularly, coined here as *multimedia-augmented services*, *microworlds*, and *multimedia frameworks* (discussed in sections 2.2 through 2.4). During this brief review, we discuss some relevant abstractions as they emerge. Section 2.5 further distills and summarizes the abstractions found.

2.1 Multimedia Authoring: The Synchronization Abstraction

Creating a presentation that contains multiple media involves a good deal of media-specific work (recording, selection, and editing of video, audio, text etc.). The true multimedia authoring is centered around the task of synchronizing such multiple-media parts into a – maybe interactive – multimedia presentation in time (parallel and sequential ordering of tracks) and space (‘on the screen’, for visual media and user interface elements). Thus, multimedia authoring can be said to be the task of spatio-temporal synchronization. A number of ways exist for specifying synchronisation such as timeline (‘left-to-right’), transition nets, space-time (considering the time as one of several dimensions in a finite coordinate space, as in MHEG). Apart from the user-centered, authoring-specific task of synchronisation specification there is the issue of *delivering* a synchronized presentation, an issue that has to be dealt with in all of the below-mentioned multimedia services but which is less emphasized in this article because we focus on the ‘what’ offered to a DMM application developer, not on how it is achieved at runtime. Time-line based specification has proven to be the most intuitive scheme for authors, whereas, e.g., PetriNet-based schemes (which can be derived from timeline-based ones) can be used as a basis for both system-level checks and runtime execution. See chapter 15 in [12] for more details.

A key precursory task to synchronization specification is the *design* of an overall multimedia presentation, based on higher-level semantics such as rhetorics, didactics, and the concepts to be conveyed with the presentation. Todays multimedia authoring systems pay very little attention to this predominant issue, so that we postpone its discussion until section 3.3, where the value of hypermedia in this context will become evident.

2.2 Multimedia-Augmented Services

Many functionalities used for some time in general-purpose computing can be adapted for specific use in distributed multimedia systems. Two domains in which this trend is most visible are communication protocols and operating systems. In order to benefit from the services discussed in this section, multimedia application developers must today rely on simple application programming interfaces, APIs. API-level programming is particularly error-prone if a large number of subsequent service calls by an applications are interrelated, as is often the case in the multimedia context. Section 2.3 and, in particular, chapter 3 revise more consistent approaches. Below we focus on the principal multimedia-specific requirements and services provided by such protocols and operating systems.

this abstraction used across many such services. QoS is meanwhile well introduced in the multimedia literature and does not have to be deeply reviewed here. Rather, we want to point out for our context that QoS processing in a typical DMM system has two complementary aspects: (a) QoS specification and negotiation, consisting of communicating performance specifications between relevant parts of a DMM system, including the user. Negotiation normally results in an agreement between all relevant components; the result is a contract stating the conditions and the performance commitment types (such as best effort, bounded, statistical, guaranteed). (b) Maintaining the negotiated quality levels during the application's lifetime, achieved through the internal resource management mechanisms of the subsystems.

As the importance of QoS for managing the system resources is well recognized, QoS-oriented DMM services and system architectures are emerging. Synchronization specifications, too, start to reflect user level QoS in the timing, ordering, and coordination of events; this user level is translated into system level QoS as part of the playout schedules.

Communication Protocols: it has become common knowledge that protocols used in todays mainstream data networks (such as TCP/IP, Ethernet, and X.25) are designed for handling non-continuous data and are ill-suited for continuous media. This is because of the different nature of continuous and non-continuous data: the former require high-rate isochronous transmission, but are somewhat more error-tolerant; the latter are more error-sensitive, but less demanding as far as, e.g., delay and jitter are concerned. Traditional protocols provide only best effort-type, point-to point communications; thus they cannot reliably support isochronous traffic, nor can they serve efficiently in multipoint situations such as teleconferencing and cooperative work. Real-time (RT) capability assures timely reaction to events such as generation of video frames by a camera. When these frames are transmitted by a protocol without RT and displayed by a non RT operating system, the display may become jerky because of unpredictable delays in transmission and decoding. With RT capability these delays can be guaranteed to be bounded. The required values are defined using QoS specifications. These in turn depend on the synchronization requirements of the application.

Given the increasing importance of multimedia, research has addressed adequate protocols. The most important requirements for multimedia-capable protocols are: (i) onfigurability in terms of QoS specifications, permitting various tradeoffs between speed, isochrony and error resilience (difficult to achieve on top of complex resource management mechanisms); (ii) coexistence of a number of different types of connections in a network; (iii) support for multipoint communications, useful in multicast and cooperative scenarios; (iv) low execution overhead (lightness), needed to reflect high data rates and smooth processing of continuous media.

Some protocols have been developed for specific layers (e.g., XTP and TP5), others in the form of entire protocol suites (e.g., Heidelberg Transport System and Tenet). They all feature a combination of different transmission modes with the possibility of resource reservation. ST-II (Stream Protocol-II) and RSVP (Resource ReSerVation Protocol) are both transport-level protocols supporting guaranteed performance over one-way multicast (point-to-multipoint) communications. See chapter 11 in [12] for a detailed description of these protocols.

There is a growing pressure to update the Internet from its original point-to-point, best effort-type service towards a multimedia-capable network. The core of Internet being the IP protocol, it is not surprising that most research effort is aimed at improving and generalizing IP, such as in Next Generation IP [7].The latter contains format provisions for expanded addressing, multicast routing, labeling flows by QoS specifications, and security / privacy. Another project aims at providing QoS capability for IP multicasting over ATM. Here, two service classes are defined: (a) real-time, with guaranteed packet delay and rate, using RSVP and ATM's connection setup and bandwidth reservation mechanism, and (b) best effort, for applications which tolerate variations of throughput.

Operating Systems: Another problem well recognized by the multimedia community is the limited value of todays operating systems, again due to the real-time nature of continuous media. Decent multimedia presentations on Unix, MS Windows, OS/2, Apple System 7, and the like, depend on the ubiquitous availability of the necessary resources. This can be easily verified, e.g., by observing the jitter and slowdown of an Apple Quicktime movie player, caused when other jobs are initiated. These problems are not due to a real shortage of resources, but to inappropriate scheduling. Continuous media processing is known to be of ‘soft real-time’ nature, where occasional computing errors or failures to meet deadlines (such as missing or delivering too early or delaying a video frame) do not cause physical or functional damage as it may happen in conventional real-time systems. Further differences include the fact that time-critical processing of continuous media is mostly periodic (e.g. reception of video frames), and that applications may adapt to variations of resource availability via QoS negociation (e.g. by decreasing the quality of video playback).

The principal requirements placed on multimedia-capable operating systems are:

- Real-time CPU scheduling. Rate-monotonic and earliest deadline-first are the most frequently used policies.
- Memory, buffer, and file management policies to support real-time scheduling. For example, pages allocated to a real-time task might be locked into physical memory.
- Support for real-time synchronization. This implies efficient inter-process communication, event notification, and mutual exclusion mechanisms.
- Possibility to process non real-time tasks in the usual manner; this implies the coexistence of real-time and traditional components within one system. A multimedia application typically entails both types of tasks, each can then be executed by the appropriate part of the OS.
- Low-overhead task management; this is needed because of frequent task switching. Lightweight threads and thread switching in user space are employed in some systems. For more details, cf. excellent discussions in [12].

Sometimes it is difficult to distinguish between the services (accessible from outside) and the underlying implementation mechanisms. Many of the services are ‘QoS-capable’ versions of standard services, e.g. real-time ports, reactive objects, real-time threads and bounded-delay RPC (see 2.5).

2.3 Multimedia Development Microworlds

By multimedia microworlds we mean application development support systems available usually as extensions to system platforms. Such microworlds (sometimes called environments or toolkits by the vendors) are rather self-contained and platform-dependent. Usually, their design and architecture is controlled by a single vendor; their ‘openness’ emphasizes the possibility to shield applications from the addition of new system-level functionality (e.g., ‘third party’ interface cards such as frame grabbers, video compression HW/SW). Microworlds are hardly designed for portability to different platforms and even less for implementing applications over heterogeneous distributed platforms. They offer abstractions at the API level, but APIs of different microworlds are largely incompatible.

Among the best known microworlds are Apple’s Quicktime, Microsoft’s Windows Multimedia Extensions MME, and IBM’s Multimedia Presentation Manager MMPM/2 for OS/2. MME provides mostly device abstractions and file services, while MMPM/2 features a more advanced form of device abstraction, where implementation details of a ‘device’ (e.g. hardware or emulated) are hidden from the programmer. These logical devices are controlled through commands in a Media Control Interface. Synchronous data streams are supported through a stream programming interface, see chapter 3 in [6]. The philosophy of Quicktime, initially an extension of Apple's Macintosh Operating System, is centered around the idea of integrating continuous media among the traditional data types. The main abstraction in Quicktime is the movie data type (which also covers audio and other media), designed to deal with time-dependent aspects: real-time and interactive playout, editing, and various compression schemes. New developments include QT-VR (Virtual Reality) and QTC (Quicktime for Conferencing); there are also efforts to make Quicktime evolve towards a platform-independent framework. Numerous vendor-specific multimedia extensions to UNIX exist, complemented by Muse from the Athena consortium. Due to heavy vendor competition, none of these is expected to play a major role in an open market.

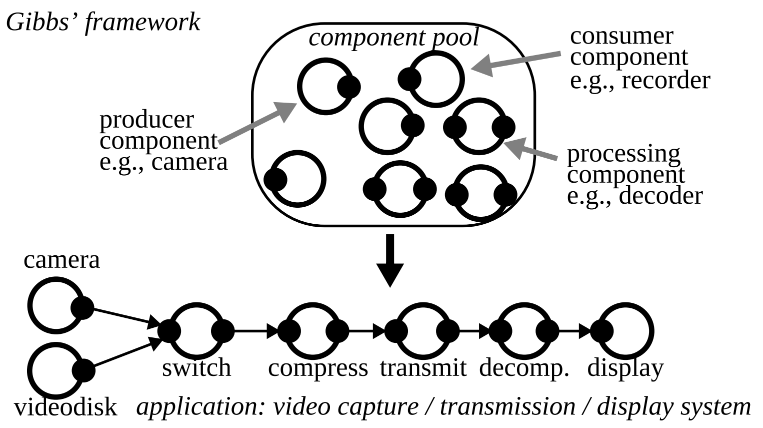
To summarize, microworlds basically act as intermediaries between applications and low-level system functions. For example, the Media Control Interface (MCI) of the Windows Multimedia Extensions is designed to support device abstractions such as VCR, videodisc, display controller, MIDI player and others. These abstractions shield the programmer from the particularities of real devices to a certain degree only.

2.4 Multimedia Frameworks

Known multimedia frameworks can not yet be considered encompassing frameworks as required in this article, but they offer a level of sophistication and consistency that goes beyond microworlds. They facilitate the design and implementation of applications in heterogeneous distributed environments. Much more than microworlds, frameworks are intended to be extensible and portable; applications developed on such frameworks are (ideally) platform- and location-independent. We will briefly discuss recently proposed frameworks based on object-oriented class hierarchies.

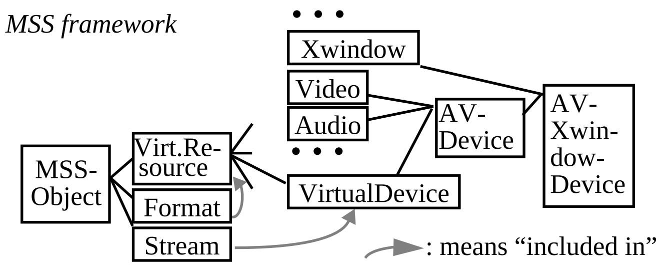
Gibbs’ framework [6] derives from four abstract classes: *Media* abstracts different media types (audio, video, image, text, etc.).*Transform* represents certain operations on media (such as compression, filtering). *Format* describes the external representation of media data for import and export. Examples are ASCII, MPEG and CIF. *Component* represents ponents have three subclasses which make up the basic types of *multimedia processing elements, MPE* (cf. corresponding sidebar): producers, transformers and consumers of media streams. Attached to components are objects called ports; pairs of ports are linked by connections, which can transmit streams. An application is created as a set of interconnected components; an initial design is refined by specializing the instances.

Multimedia Frameworks

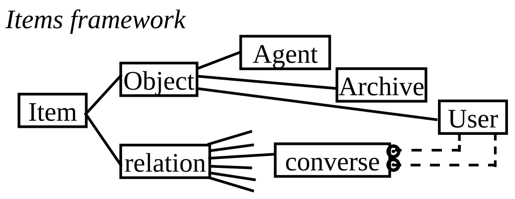


Gibbs' framework
component pool
consumer component e.g., recorder
producer component e.g., camera
processing component e.g., decoder
camera
videodisk
switch
compress
transmit
decomp.
display
application: video capture / transmission / display system

Different multimedia frameworks use different basic types of *multimedia processing elements, MPE*, and of *time-related media*, all usually integrated in an object-oriented class hierarchy. In Gibbs' framework, only three basic types of MPE are distinguished, called components. In most known multimedia frameworks, the MPE types reflect technology-bound aspects of media processing. Almost all of them use system-controlled streams to 'plug together' MPEs.



A small excerpt from the class hierarchy of the MSS framework, representative for most known class hierarchies of object-oriented multimedia frameworks, shows that such hierarchies usually reflect technology (cf. 'X window') and conventional media types (cf. audio, video).



An excerpt from the Items class hierarchy illustrates the high, media-independent level of semantics and the distinction between objects and relations. The six relation subclasses match all possible combinations of the three subclasses of 'object'. E.g., 'converse' relates users among one another and may be instantiated e.g., as a videophone stream or an email connection.

The scope of the Interactive Multimedia Association's framework called Multimedia System Services (MSS) [8], is wider and somewhat more vague than in the previous example. The stated purposes include to 'provide abstractions that make it possible for applications to deal with media devices without regard to specific characteristics of the platform, attached devices, or the network(s) connecting the platforms and devices', and 'to provide a standard methodology, especially for handling live data'. MSS supports distributed objects by complying with the Corba standard discussed in 3.2, in particular to Corba's Common Object Services Specification. Basic classes include virtual devices and media streams. Virtual devices may be processors, files or hardware devices; they have attached ports with associated formats. Applications are constructed in a plug-and-try manner within a reference architecture. Virtual devices and user-defined applications make up the basic types of multimedia processing elements here.

Sidebar 'Multimedia Frameworks' illustrates how virtual devices are imbedded in the MSS object class hierarchy, showing that the semantic level of the objects is still related to specific hardware and specific media used. The excerpt of the class hierarchy proposed in the Items framework [10] comprises very 'abstract' objects. E.g., user interfaces (class 'user') can be interconnected via a relation 'converse' which can be instantiated at runtime as, e.g., an AudioVideo stream or an asynchronous email connection. This simple example shows that much higher-level semantics of applications may be of interest than those emphasized by most multimedia frameworks.

A large number of frameworks have been developed and proposed recently (cf. [13] pp. 10-76). Although most of them are explicitly designed to be open-ended, at present no single framework is sufficiently general to cover all aspects of works. Almost all of them propose again a small set of basic types of multimedia processing elements and of time-dependent media similar to the ones discussed.

To summarize, a framework can be regarded as a pool of components which represent multimedia processing elements (MPE) plus a class hierarchy for MPE and basic media types; applications are built by selecting and interconnecting components from the pool. However, in order to be workable, this intuitively appealing approach needs a large component pool to begin with, and the facility to add new components. Most present frameworks offer MPEs and class hierarchies which reflect the ‘lowest’ level of semantics only. Details in the design of the class hierarchy differ a lot and a good rationale for this design is not yet commonly accepted. The example of the Items framework shows that it is desirable to support a choice among different technology-bound objects, even at runtime, for realizing a given ‘high-level’ object. This selection may imply a replacement or transformation of media and requires very careful design of the technology-bound class hierarchy together with higher-level hierarchies.

Another important observation concerns the ‘configuration’, i.e. the way in which the overall topology of an application (out of MPEs and streams) is described. Usually, frameworks offer a graphical depiction and description language for specifying one specific and static configuration per application as a network of MPEs. In contrast to this low level of sophistication, many approaches to the specification of configurations of interconnected entities exist, with arbitrary numbers of components and dynamic changes. These approaches have been developed in the context of a variety of domains such as distributed programming, graph theory, and hypermedia. This issue will be resumed in 3.3.

2.5 Summary of Abstractions

Section 2.1 showed that *synchronization* specifications represent the essential abstraction for multimedia authoring. In the context of multimedia processing, *quality-of-service* (QoS) was the first general abstraction we could identify. The time-related nature of media data has lead to many proposals for abstractions. The most common agreement seems to concern the transfer of data between components where the notion of *stream* is used almost throughout; e.g., protocols provide various combinations of QoS-controlled point-to-point and multipoint connections (e.g. isochronous, connection-oriented, connectionless). While synchronization, streams, and QoS are thus commonly accepted as necessary abstractions, details about their functionality and representation are still widely disputed.

Much less agreement exists about abstractions necessary for the components that make up a multimedia application apart from streams. We recall that operating systems provide low-level real-time enhanced services via ports, procedure calls, and threads. Microworlds provide services as API-level primitives directly accessible to the user or application programmer (within the scope of a given platform). Multimedia frameworks provide services usually as class hierarchies aimed at easing portable distributed application developmen, representing a step toward encompassing frameworks.Virtually all multimedia frameworks propose some kind of multimedia processing elements and, of course, of time-related media as components. Modeling these as object-oriented classes is simple at a first, an open issue at a second glance: (i) it requires adding a notion of (‘soft’ real-)time to the object model; (ii) the design rationale for such a class hierarchy is not commonly agreed; e.g., the distinction of MPE and media types ‘smells’ like the process/data distinction which object-orientation is supposed to overcome; (iii) the support of different levels of semantics is even less understood. Since point (i) above is undoubted, we can in any case retain that *time-related object* is a valid abstraction. The top-level distinction of MPE and media, however, is disputable: e.g., a video object may be a stored sequence of frames (media) or a video-generating program (MPE). Considering time-related object as a key abstraction, one may argue that further aforementioned abstractions realize this abstraction to a certain degree (and thus do not need to be separate items on a list of key abstractions to be retained):

- Real-time ports which are similar to standard OS ports in that they serve as end-points for bindings, yet time-related via QoS parameters. Rtports implemented in Chorus (a real-time OS kernel) serve as support for reactive objects.
- Bounded delay remote procedure call (RPC), extending standard RPC with guarantees for maximal delays between (remote) emission of the RPC and reception of the result (used for object invocation in ANSAware [4]).
- Devices and virtual devices as abstractions of physical multimedia-related devices and software modules that handle streams. Devices can be classified into sources, sinks and transformers. Device interfaces have stream, operational, device-dependent and device-independent components.
- Reactive objects [2], proposed as extensions to the ODP framework (cf. 3.2) in order to accommodate multimedia requirements. A reactive object (as opposed to ‘standard’ objects) can be thought of as an automaton with real-time behavior which reacts to incoming events within guaranteed delay bounds.

this, we can summarize five key abstractions to be retained, out of the great number of service abstractions found in the multimedia programming context. While these five abstractions are still not entirely orthogonal, we regard the list as a considerable step forward from the existing literature where abstraction have much less mutually exclusive meanings; in some cases the same notion is named differently by different authors, in others they are hierarchically related. The following five abastractions are much more orthogonal and still stay in line with known abstractions, i.e. do not 're-invent the wheel'.

1. *Synchronization:* synchronization specifications determine details of synchronized multimedia presentations. They must be transformed into playout schedules for all services involved. MODE offers an advanced timeline based approach, considers distribution issues, accommodates both multimedia-augmented and traditional services, and spatio-temporal aspects. Currently mainly an authoring issue, it should be harmonized with DMM programming.
2. *Stream:* Streams and stream interfaces are widely used abstractions proposed in different contexts: communications protocols, operating system support, and application design [2], [6], [8]. Streams transmit continuous media data. Stream interfaces are sources and destinations of streams, attached to other components. They complement the standard operational interface between objects. Parameters of streams / interfaces may involve QoS guarantees. Future streams should harmonize isochronous and asynchronous transmission, multihop and multicast capabilities.
3. *Quality-of-Service:* QoS controlled bindings [2] serve to establish and manage links between interfaces and can be considered as objects capable of controlling links at given QoS levels. Reactive objects and streams can be used as the design centers for QoS. However, for more sophisticated QoS handling which covers different levels of semantics from user to system level, QoS must be integrated within an encompassing framework.
4. *Time-object:* most multimedia frameworks distinguish two basic kinds of objects: multimedia processing elements (MPE) and media. The essence in terms system support is the addition of time to the object notion. Current deficiencies include: technology-bound semantics without matching higher levels, insufficient design rationale; top-level distinction of (blurring) MPE and media; streams either disguised as MPE or hidden from developers.
5. *Configuration:* as discussed in section 2.3 and the corresponding sidebar, the introduction of MPE and streams brings up the concern of 'plugging them together' into an application. Such configurations are usually static and deterministic and do not reflect the state of the art, e.g., in distributed programming.

In spite of the great variety of forms they take, all services and abstractions have the same purpose: to ease the conceptual understanding and the development of distributed multimedia systems. Todays independently developed, largely inconsistent, overlapping abstractions make the task of DMM design messy and difficult. To ease this problem, we need more powerful systems of consistent service abstractions spanning all stages of the design process and harmonizing multimedia development with distributed application development. Table 1 below summarizes the five key abstractions proposed, anticipating a few requirements that will become more evident in chapter 3.

TABLE 1. Key concepts to be retained

abstraction	**basic meaning and requirements**
1: synchronization	spatio-temporal arrangement of presentations. To be intuitive, entity-conserving, distribution-proof, suited for: unlimited and unknown durations, with flexible degree of synchrony, translation into playout schedule
2: QoS	'fuzzy', e.g. statistical description of (/constraints on) parameters of service delivery; reflects tolerance in human perception, enables resource mgmt and tradeoffs between conflicting goals.User-level: to be intuitive & controllable; all levels: to be compatible, negotiable, accessible for system-wide optimization.
3: stream	communication channel with real-time guarantees and continuous transmission of 'samples' without application intervention. In future to be multihop, multicast, covering both iso- and asynchronous transmission.
4: time-object	umbrella for multimedia processing elements (MPE) & time-related media; to be system-supported yet user-extensible & distribution-proof; to support high-level semantics which make technology, the MPE / media distinction, and run-time media choices transparent; to support QoS and synchronization.
5: configuration network	composition of MPEs, streams, media into an application; to support dynamically changing configurations with arbitrary numbers of components. To support distribution, authoring & programming

3 Distributed Application Development

In this section we survey relevant approaches to the development of complex distributed applications and their extensions towards multimedia. In distributed application development environments, the developer's work is enhanced with a consistent set of services and libraries, development languages (or language extensions), and facilities for interaction and interworking over heterogeneous platforms.

Apart from the general distinction of distributed authoring and distributed programming, distributed application development environments can be classified according to the mental model, or paradigm they are based on. We survey the remote procedure call (the basis for most client-server systems) and distributed object-oriented paradigms for distributed programming, and the hypermedia and open document paradigms for distributed authoring.Within each category we address two questions: (i) Which extensions towards distributed multimedia (DMM) support have been proposed? (ii) How powerful is the paradigm with respect to full-fledged DMM support?

3.1 Client-Server / Remote Procedure Call Paradigm

Remote procedure call (RPC) is an extension of the well-known procedure call mechanism and is directly related to the client-server notion. Interface definition languages complement different programming languages, enabling compiler-generated code (stubs) which marshalls the remote calls, interacts with the server and assures well-defined error semantics in case of failures. DCE (distributed computing environment), promoted by the OpenSoftware Foundation, is based on a specific RPC mechanism enhanced with some useful services, such as naming and authentification. DCE represents the most wide-spread basis for heterogeneous client-server oriented distributed programming environments [11].

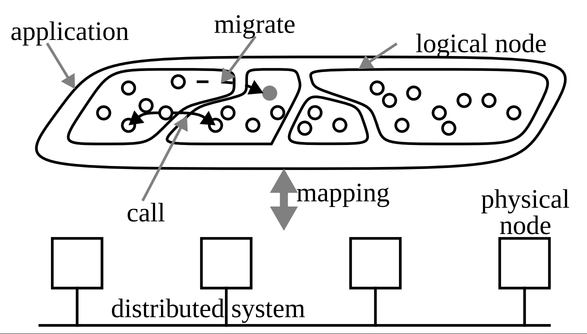
Known multimedia extensions: Extensions of RPC towards bounded delay have been mentioned in section 2. RPC extensions for mass data transfer (and multimedia data transfer in particular) have also been proposed. The most common solution is to use RPC for stream setup only and to realize the actual isochronous streams without program intervention in the runtime system. Multimedia extensions to DCE in particular have been investigated [1].

Expressive power and suitability of the paradigm: There is both a technical and a conceptual mismatch between the nature of distributed multimedia processing and the nature of RPC. Technically, RPC is optimized for command-response type communication and short messages. Therefore, any kind of efficient mass data transfer requires that RPC be complemented by further mechanisms. Conceptually, RPC imposes client-server relations on distributed software components; this kind of relation is already questionable for ordinary distributed applications (many real-world entities are not purely client-server-related); in particular, it matches badly with conversational uses of multimedia. The same is true for RPC as a communication mechanism: it turned out that RPC cannot be regarded as a transparent extension of local procedure call, since asynchrony and error semantics have to be introduced for decent distributed programming, dramatically changing programming styles in comparison with ordinary procedures.

In summary, the RPC paradigm does not seem to be a good candidate in our search for encompassing multimedia paradigms. It was included in this survey mainly because some of the distributed object-oriented services discussed below have been built on top of, or integrated with client-server-oriented services, DCE in particular.

3.2 Distributed Object-Oriented Paradigm

DOO: distributed object-oriented programming



Distributed object-oriented programming may support distribution transparency: local and remote method-calls to other objects need not be distinguished by programmers; the mapping of the fine-grained objects onto logical / physical nodes may be determined at runtime, objects may even migrate to other nodes dynamically. Remaining drawbacks in the multimedia context: the connectionless model of inter-object communication does not match easily with streams, and distributed authoring is not considered.

ject equals data plus methods i.e. data-manipulating code, data accessible through methods only), powerful type concept (classes, inheritance), and extensibility (polymorphism and late binding). Its message-based method invocation realizes connectionless inter-object communication. The distributed object-oriented (DOO) paradigm is much less established, since different lines of thought emphasize different aspects (such as heterogeneity of platforms and languages, location-transparent method invocation and object migration). The state of the art in DOO services is presently pushed by a competition of industry standards, the most relevant of which are Corba by the Object Management Group, COM by Microsoft and DSOM by IBM. Others like DOE (Sun), DOMF (HP), and PDO (Next) are excluded here for brevity. The industry efforts are complemented by ISO ODP [11] and by academic DOO programming languages.

All industry efforts extend object-oriented programming across heterogeneous computer networks and across different programming languages. Corba was specifically designed to be distributed, several Corba-compliant implementations exist. IBM's SOM and Microsoft's COM are rather proprietary and competitive object-oriented frameworks, Distributed SOM (DSOM or SOM2) introduces DOO concepts in a Corba-compliant way. It realizes some advanced features which can otherwise be found only in academic DOO approaches, in particular location-transparent method calls. The ISO ODP (open distributed processing) model [11] may be considered less object-oriented. ODP offers five different views on the software under development. The most important one for software design, called computational viewpoint, uses the term object to denote a communicating process. Since ODP is a model only, further discussion will refer to the concrete implementation ANSAware. There, the computational viewpoint defines ‘objects’ as processes together with interfaces which define a set of callable operations plus invocation constraints.

Known Multimedia Extensions: ANSAware has been extended for multimedia [4]; in this approach, the basic object/interface concept of ODP and ANSAware has been left largely unchanged. Instead, two system-provided object categories were introduced for virtual devices and streams, pretty similar to the corresponding concepts introduced in chapter 2. The limited conformance of ODP to the DOO paradigm and limited general acceptance restrict the overall value of this interesting approach. A rather academic, but conceptually very interesting approach must be mentioned here: the MODE system (cf. [9], pp. 207ff.). MODE is truly DOO-based and combines DOO distribution transparency with QoS negociation. Streams are entirely hidden from the programmer. Rather, application-specific media object classes are derived from system-defined classes, giving the runtime system a possibility to access application data. Based on these and on information about resources, data locations, network topologies etc., MODE automatically computes an optimal path from information sources via transformers to media presentation (or again storage) devices, sets up necessary streams, negotiates QoS with the underlying services and triggers execution, all following maybe a single application-level method call. MODE even supports multimedia authoring via spatio-temporal synchronization specifications. The MODE research project is discontinued however, and the lack of standards-compliance and openness (with respect to programming languages) let it stay an academic approach.

Expressive power and suitability of the paradigm: The simplicity, rigid encapsulation, and general applicability of object-orientation are the foundations for its success. DOO - in contrast to RPC - supports flexible distribution of the fine-grained objects at run-time. Nevertheless, the ‘ultimate’ solution for DMM application development, harmonizing the four fields discussed throughout the article, needs a considerable revision of the DOO concept. Two problems will be discussed in chapter 4 together with possible solutions. They concern the adequate representation of streams in DOO and the harmonization with distributed authoring. At this time, it is to be retained that DOO is an excellent starting point for our search for an encompassing paradigm and framework. It must also be mentioned, however, that DOO standards like Corba are well advanced and hard to change as fundamentally as full multimedia support would require.

3.3 Hypermedia Paradigm

The hypermedia paradigm is centered around the following four basic kinds of elements: *nodes* (meaningful units of information), *links* (relations of any kind between nodes), *anchors* (selections within nodes from and to which links lead), and *webs* (coherent sets of nodes and links, also called hypertexts or hypermedia documents), plus various composites such as versions, views, subsets, trails, etc. The term hypermedia stresses the support of different media in different nodes, the underlying paradigm is sometimes still called hypertext (hypermedia will be used in the remainder). The hypermedia scene is split into an open, distributed, but (relatively) low-featured, and a sophisticated but proprietary part. The former comprises the open standards, mainly MHEG and HyTime, plus the Internet web approaches, mainly WWW and Hyper-G. The latter consists of a myriad of systems built for academic, sometimes commercial use; they show a trend towards accepting the so-called Dexter Reference Model [3] as a common reference.

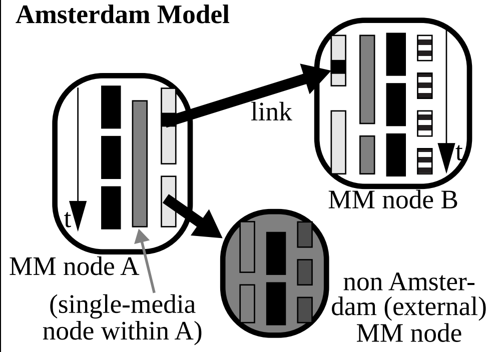
edited anymore (cf. CD-ROM). The final form concept makes MHEG ill-suited as a basis for very general solutions which we search. This statement remains true even though MHEG has lately undergone radical changes in the light of Interactive TV applications. Recent comparisons have brought evidence to MHEG's very high conceptual overload which makes it very cumbersome to author MHEG documents (cf. [13], pp. 140ff.). HyTime is an extension of the ISO standard graphics markup language, SGML. Unlike MHEG, it does not include an object model, nor specific presentation rules. The concepts of hypermedia document and node are intertwined in the HyTime document concept, and the elaborate multimedia synchronization (specification) support is not distribution-proof. Thus, HyTime is not a good starting point either.

WWW is based on HTML, a markup language which is an SGML derivate like HyTime. It is well integrated with the Internet communication architecture. Like HyTime, WWW does not reflect some of the hypermedia principles defined in the Dexter Reference Model. E.g., all link information is embedded in the nodes instead of a separate database, making it is almost impossible to grasp and visualize a hypermedia document as a distributed ‘web’. WWW does not model or support synchronized multimedia and usually uses non multimedia-capable Internet protocols. Due to a rudimentary anchor concept, to date only some of the media may be sources of links. While continuous media are supported in WWW, presentation is delegated to ‘viewers’. Synchronized multimedia presentations are thus out of scope. While the upcoming possibility to include (Java) programs into WWW documents makes ‘everything’ possible in WWW, the Web is more of a collection of possibilities then a coherent system based on an elaborate paradigm. Hyper-G offers remedies to some of the WWW syndroms: it keeps a separate web database and features consistent update of link destination changes, supports access rights, some degree of ‘context of use’, and versioning. Hyper-G / WWW cross-compatibility is emphasized. To date, Hyper-G still lacks some desirable features like user-defined anchor models, an elaborate type concept, and further advanced features discussed below.

The Dexter Reference Model goes beyond the state of WWW. Its most important contribution is the clear distinction between web database, node contents, and user interaction. Quite a number of Dexter-compliant implementations have been discussed in the literature [3], and so-called ‘second-generation’ variants support user-defined node / link types.

Known Multimedia Extensions: While all the above-mentioned systems support multiple media, true multimedia features such as integrated synchronization models, anchors for video or audio elements, and support for isochronous distributed communication are restricted to a few academic approaches. The most important one is the extension of the Dexter Reference model towards multimedia authoring, called Amsterdam model ([3], pp. 50-62). This model harmonizes distributed and multimedia authoring by extending the node concept towards synchronized multimedia.

Amsterdam Model

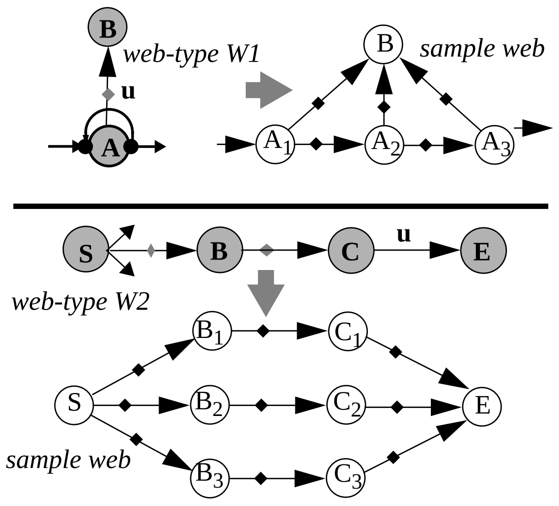

link
t
t
MM node B
MM node A
(single-media node within A)
non Amsterdam (external) MM node

The Amsterdam Model combines multimedia synchronization and hypermedia. With Amsterdam-based synchronized multimedia nodes, embedded single-media nodes keep their identity and can be used in other contexts, e.g., in other multimedia nodes. Links can point directly to the embedded nodes, and the author has to interface a single system only. Without Amsterdam, multimedia presentations can only be imported as external nodes, created by an external multimedia authoring tool; embedded nodes can neither be reused in other contexts nor directly linked.

Expressive power and suitability of the paradigm: In spite of the shortcomings of present realizations, we rate the expressive power of the hypermedia paradigm as very high. In particular, the top-level distinction of nodes-and-links is well suited for expressing higher-level application semantics as so-called semantic networks, such as rhetoric or didactic concepts, design histories of complex systems, or whatever the subject matter of an application (domain) may be. Means have been implemented for mapping different levels of semantics onto one another (yet realizations have mostly been ad-hoc up to now). As a disadvantage, the paradigm lacks an elaborate type concept, but in this area the DOO paradigm has its major strength. Experience from the Nestor project [9] backs all these arguments very strongly. Chapter 4 will discuss a combined hypermedia-DOO paradigm as a foundation for an encompassing framework.

A less recognized yet important field of research is the support for ‘families’ of webs, discussed mainly in literature about formal hypermedia models. We call such families ‘web types’ and cite PreScripts ([9], pp. 273ff.) as an example sically, allowed and forbidden ways of interconnecting them). Tutorial, bug report, and flow chart are examples from the endless list of possible web types. Since nodes and links form graphs, the elaborate research about graph grammars may be consulted here, but their value in the hypermedia context is limited and has already been reflected in existing approaches to web types. In a world where the distinction of media and processing elements becomes blurred, web types might obviously be used as well to describe the configuration abstraction in a flexible way, an issue that had been left open in chapter 2. This fact adds to our arguments that multimedia / distributed programming and authoring should be harmonized in an encompassing paradigm and framework. We will resume this discussion in chapter 4.

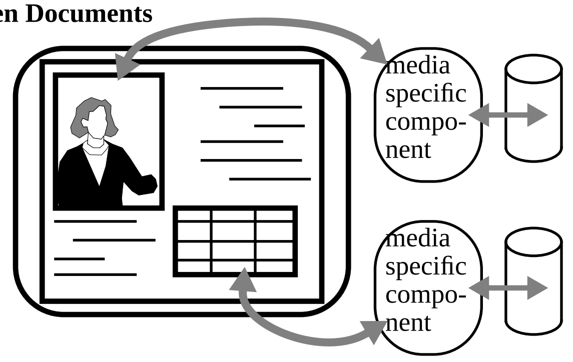
Web Types



Web types have been introduced in order to describe 'families' of hypermedia documents. The simple examples shown here use a web type concept called PreScript which allows to distinguish optional and mandatory parts of hypermedia documents, sequences of nodes (cf. node A in type W1), and bunches of links (cf. the link starting from node S in W2). In the example, the unique-destination-flag **u** indicates links which, in an actual web of the type defined, are inbound to a single node. Web types can be used for defining higher-level semantics of hypermedia documents, for programming reusable navigation support in hypermedia, and for assuring common-look-and-feel of hypermedia document families. In an encompassing paradigm as discussed in the article, they may equally serve for specifying dynamic configurations.

3.4 Open Document Paradigm

Open Documents



Open document models are usually implemented using a DOO base system. They support the inclusion of remote parts (in different media) in a hub document. The UI centered handling of open documents is very authoring-centered, yet synchronization is rarely supported. The hub-component structure is less powerful than hypermedia webs. Open document systems excell with component-based programming support based on the rigorous standardization of the DOO system, class library, and set of methods / events each part has to serve.

Simply put, this paradigm allows different distributed tools to work on a distributed document. Again, two major camps can be identified: industrial de-facto-standards which overlap largely with the industrial DOO approaches, and ISO standards. The latter are either based on SGML (see 3.3) or on the Office Document Architecture and offer basically a subset of the functionality discussed in 3.3; they will not be treated further. Treatment of the industry efforts will focus on their contribution beyond DOO and hypermedia: in particular, they offer true interoperability of components. We already mentioned the need for integrating many heterogeneous, in part pre-built software modules in a large software system. This problem space is often designated as component-based programming, and is much aggravated in the multimedia context. While open DOO approaches lay some ground for the interoperation of heterogeneous components, they do not determine the context of interaction rigorously enough in order to support easy combination of modules which were developed without 'knowing each other'. The industrial standards for open documents put an emphasis on this aspect.

Due to their market relevance, two industrial standards seem to receive most attention: OpenDoc, a SOM-based multi-vendor effort, and Microsoft's COM-based OLE. An open document standard can be considered to consist of three major vices beyond basic interaction needed for making components interoperable - easier achieved if restricted to a certain application domain (here: document processing), and (iii) predefined functionality offered to component developers (e.g., in the form of prebuilt class libraries).

Known multimedia extensions: To date, open document frameworks encapsulate time-based media with the notion of 'playable object'. Such objects are subordinate to a text-based hub document. Harmonization with multimedia authoring similar to the Amsterdam model mentioned in 3.3 is in its infancy only.

Expressive power and usability: The open document efforts show that more than the DOO 'groundrules' is necessary in order to allow for distinctly developed objects to interoperate within a common user interface. The paradigm as such has major drawbacks which make it seem inappropriate for use as a general multimedia application paradigm: (i) Hub documents are usually text-based and associated with a page-based appearance. (ii) The information modularization concept 'components of a common document' is insufficient. (iii) The application modularization concept 'components which handle a specific document part type each' is equally insufficient. (iv) Document-centered application development is usually based document flows, an approach which has been found insufficient in many publications about workflow management and cooperative-work [10]. It seems close to impossible to extend the document paradigm towards a general power and expressiveness comparable to that of DOO or hypermedia. Therefore, the challenge comes up to map the merits of open documents (component-based programming and interoperability) onto other paradigms.

3.5 Summary of Contributions

Table 2 lists the major advantages of the paradigms discussed in chapter 3. Since RPC has shown limitations in the multimedia context and since its advantages are a full subset of what the DOO paradigm offers, RPC is not listed below. Since RPC and DOO contribute to distributed programming and hypermedia and open documents contribute to distributed authoring *by design*, respectively, this fact is also left out, even more so since we intend to harmonize contributions and requirements from all four fields involved. It should be retained, too, that the important contributions to component-based programming made in the context of the open document paradigm are not particularly bound to that paradigm and that the paradigm as such is rather inappropriate for our purpose.

TABLE 2. Major contributions made by the relevant paradigms of distributed application development

paradigm	**contribution**	**keywords**
DOO	object-orientation	encapsulation, inheritance, polymorphism; suits MM frameworks, components, distribution
	dist. transparency	location-transparent calls, object migration, system-assisted configuration / QoS mgmt.
hypermedia	nodes & links	accommodation of semantic networks (levels of semantics), unification w/ synch. model
	web types	type concept for distributed authoring, may serve for configuration semantics as well
(open doc.)	components	DOO base system, class library, required set of methods / events / UI elements per part

4 Conclusion: Towards An Encompassing Paradigm & Framework

Trying to summarizing all of the above chapters in two phrases, we can state that:

- Multimedia application development is still in its infancy, requires five major abstractions, and is currently best supported with multimedia frameworks - most of which are incompatible with frameworks for distributed programming; programming and authoring (synchronization in particular) have yet to be integrated.
- Distributed application development is also split into a programming and an authoring world, best supported by the DOO and the hypermedia paradigm; the latter are complementary; both have been used in feature-rich open distributed frameworks; these paradigms are extensible and have in part been extended towards multimedia.

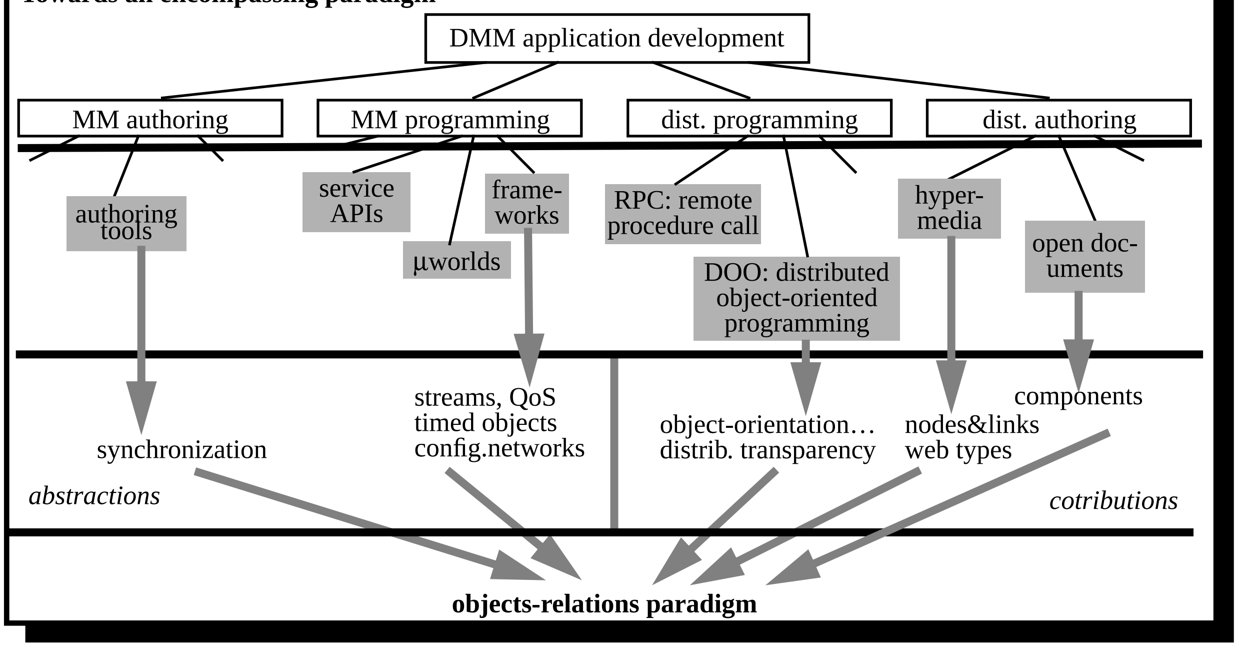
Summarizing the most interesting contributions discussed in this article, we may retain that:

- Gibbs' framework has applied a rigorous *object-oriented* approach (although not truly DOO) to most of the multimedia programming abstractions (points 2-5 in table 1, on a low level of semantics however).
- MODE has applied rigorous DOO to multimedia, supported synchronization (without true conceptual integration), but did not make streams user-controllable and omitted distributed authoring;

- Items showed that higher-level semantics for timed objects and streams are highly desirable.
- PreScripts supported web types in a way that may be extended towards dynamic configuration networks.

As illustrated in sidebar ‘towards an encompassing paradigm’, it is an appealing idea to integrate the abstractions required in the context of multimedia services with the contributions made in the context of distributed application development frameworks. Since the PRC paradigm has been disregarded and the contribution of the open document paradigm is not bound to that paradigm, the analysis points at an integration of the DOO and hypermedia paradigms.

Towards an encompassing paradigm



4.1 Why Integrate?

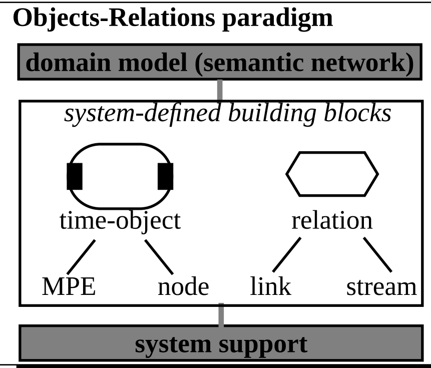
Recent discussions about lean programming have created awareness of the dangers of integration. Feature-ladden systems have become unmaintainable and memory-devouring. Our argument is not in favour of a monolithic system and of a complicated underlying model, but in favour of a simple yet powerful paradigm, modular yet harmonized components, and sophisticated component-based programming. Some arguments in favour of this approach are listed below.

- As MODE has shown, a system with consistent access to all relevant (application and system) information can provide sophisticated support (here: for network-wide QoS negociation, auto-configuration and auto-distribution).
- As the Amsterdam example has shown, the harmonization of concepts (here: of the hypermedia paradigm with the synchronization abstraction) can lead to more functionality than the sum of the parts would represent.
- Today, a DMM developer would have to use, for example, Macromedia Director™ for synchronization, WWW for embedding in a distributed hypertext, C++ and Microsoft MME (or at best an academic-stage framework) for realizing advanced multimedia functionality, and Corba for making the solution a true open distributed program (note that Java is no alternative - yet? - since it allows program distribution but not distributed programming). MME would, e.g., drastically restrict the openness and make distribution-transparency impossible. It would be cumbersome to write such an application and very difficult truly integrate the parts mentioned.
- As distributed programming frameworks are much more advanced, developing incompatible multimedia frameworks is insane. Distributed programming and distributed authoring become blurred, but Corba and WWW coexist.

4.2 How to integrate?

While the harmonization of the four fields mentioned remains a substantial research and development issue, this article has pointed out the major milestones and ingredients of a harmonized solution. Sidebar ‘objects-relations paradigm’ provides a glimpse onto a unification of DOO objects, a time concept for objects, and hypermedia ‘nodes and links’.

a PreScript-like ‘we type’ approach. System-wide synchronization can be integrated following the Amsterdam approach. Streams (and asynchronous channels) can be harmonized with hypermedia links, and media (as hypertext nodes) and MPE truly become blurred at the level of semantics considered here. Since objects and relations are distinguished at the top level, both sophisticated system support (consistency checks, autoconfiguration and autodistribution, bindings, QoS negociation) and ‘upward’ mapping onto application semantics (semantic networks) become feasible. Thus, the objects-relations-paradigm serves also as a binding element between the system (technology) level and the application semantics. Finally, a rigid standardization of required methods / events for time-objects and relations may build the first step in a effort to provide component-based programming support similar to the one offered by open document systems.


Objects-Relations paradigm
domain model (semantic network)
system-defined building blocks
time-object
relation
MPE
node
link
stream
system support

In a model that combines DOO and hypermedia concepts, hypermedia nodes and multimedia processing elements (MPE, including virtual devices) are unified as time-objects. Streams and links are both modeled as relations, with advantages that have in part been discussed with the Items approach. The design here illustrates that in an encompassing paradigm, an MPE can equally well create a stream to another MPE or a link to media data. Vice versa, ‘web surfing’ may involve MPEs transparently. Altogether, time-objects and relations can be subject to synchronization and to QoS optimization. Objects-relation networks can be mapped ‘upward’ onto application specific semantics and ‘downward’ onto technology.

4.3 Outlook

The harmonization of the four fields of DMM development requires further research, yet this article has shown that important contributions exist, paving the way towards a unification of the hypermedia and DOO paradigms and towards encompassing frameworks. In order to really reduce the complexity of distributed multimedia application development, any effort must support the breaking-up of concerns into small, manageable pieces in order to avoid the syndroms of fat software. Object-orientation and component-based open documents represent a good starting point here.

Acknowledgments: This work was partly supported by Grant No. 4566 of the National Science and Engineering Research Council of Canada**.**

5 References

1. P. Adcock, N. Davies, and G. Blair, “Supporting Continuous Media in Open Distributed Systems Architectures,” in DCE - The OSF Distributed Computing Environment, LNCS 731, A. Schill (ed.), Springer Verlag, Berlin Heidelberg, 1993, pp. 179-191.
2. G.S. Blair, G. Coulson, M. Papathomas, P. Robin et al., “A hybrid approach to real-time synchronisation in distributed multimedia systems”, Lancaster University Distributed Multimedia Research Group Report No. MPG-94-21, 1994.
3. Communications of the ACM (special issue on hypermedia), Vol. 37, No. 2, February 1994.
4. G. Coulson and G. Blair, “Meeting the Real-time Synchronisation Requirements of Multimedia in Open Distributed Processing,” Distributed Systems Engineering Journal, 1994.
5. A. Danthine and O. Bonaventure, “From Best Effort to Enhanced QoS,” in Architecture and Protocols for High-Speed Networks, (O. Spaniol, et. al., ed.), Kluwer, 1994.
6. S.J. Gibbs and D.C. Tsichritzis, “Multimedia Programming - Objects, Environments and Frameworks”, Addison-Wesley, 1994.
7. R.M. Hinden, “IP Next Generation Overview”, Internet Draft, October 1994.
8. Interactive Multimedia Association, “Multimedia Systems Services,” Draft Recommended Practice, 1995.
9. M. Mühlhäuser (ed.), “Cooperative Computer-Aided Authoring and Learning”, Kluwer, Boston, Dordrecht, London, 1995.
10. M. Mühlhäuser, “Modeling and Design of Complex Cooperative Software,” Proc. 1st IEEE Conf. Engineering of Complex Systems, Ft. Lauderdale, FL, Nov. 6-10, 1995.
11. J. Nicol, C. Wilkes, and F. Manola, “Object Orientation in Heterogeneous Distributed Computing Systems”, IEEE Computer, June 1993, pp. 57-67.
12. R. Steinmetz and K. Nahrstedt, “Multimedia: Computing, Communications, Applications”, Prentice Hall, 1995.
13. International Workshop on Multimedia Software Development, Berlin, March 25-26 1996, IEEE Computer Society Press, 1996