A Modeling / Programming Framework for Large Media-Integrated Applications

Max Mühlhäuser
Computer Science Dept., Institute for Telematics, Telecooperation Group
University of Karlsruhe, D-76128 Karlsruhe, Germany
max@tk.telematik.informatik.uni-karlsruhe.de

1 Introduction

Most present multimedia applications represent selfcontained off-the-shelf tools for specific tasks. We believe that in order for multimedia technology to gain widespread use, *media-integrated* applications must be emphasized; such applications integrate customized multimedia use with 'conventional' applications, using enterprise workflow models as the embracing concept.

Today, however, the development of large and cooperative media-integrated applications is a painful and cumbersome crafting task, due to the lack of an adequate software technology. As a consequence, we propose the development of a more adequate software technology which we coin as the move from object-oriented to *items-oriented* programming. Cooperative media-integrated applications will rapidly gain importance in the future for the following three reasons.

Cooperation trend: multimedia base technology evolves rapidly and is marked by a tight integration of three large market segments: consumer electronics, information technology, and telecommunications. Example elements of this base technology include ATM switches and networks, 'multimedia-proof' communication protocols (ST2, XTP, ec.) and storage technology (CD-ROM extensions, document architectures, databases), media servers in distributed systems, compression standards like JPEG and MPEG, and videophone standards like the H.200-series. Many existing multimedia *applications*, however, represent selfcontained off-the-shelf tools for very specific tasks (`authoring' systems for multimedia presentations, videoconferencing tools, etc.).

This is in drastic contrast to the *integration* trend which marks many software application domains, such as Computer-Integrated Manufacturing and Engineering (CIM, CIE), and office automation. Such integration, in turn, is viewed as the base for better *cooperation* support in an organization; cooperation, here, has two facets:

- *Workflow-type cooperation:* organizations try to make better use of their large enterprise networks by integrating their formerly isolated application packages; thereby, outputs of packages are fed into other ones in an intelligent way, packages mutually call one another, and, most important, superimposed workflow management software manages and traces the flow of work in the organization.

- *CSCW-type cooperation:* computer-supported cooperative work -CSCW-, together with workflow management, is supposed to lead to enhanced cooperation of all parts of an enterprise (supply chain, production process, etc.); with CSCW-type cooperation, the focus is on the *humans* involved.

Media integration demand: in the context of this integration / cooperation trend, multimedia is no more an end in itself, but a requirement. Workflow-type cooperation requires *entire* activity records and documents to be computerized (traditional database applications kept only short, abstract descriptions: database records). This leads to the use of multimedia objects such as scanned images, digitized audio annotations, video and audio conference recordings, summarized as *persistent media*. CSCW-type cooperation, on the other hand, requires *transient* media to be captured, transmitted and presented in real time (video / audio conferencing, application sharing, etc.).

Software technology requirement: A look at current software technology shows that traditional software development tools are totally inadequate for large distributed applications. E.g., the majority of CASE environments focus on sequential software intended to run on a single computer. As to distributed applications, there is a strong focus on client/server- (and, along with this, RPC-) based techniques; strict client/server-structures, however, do not match very well with the complex, irregular organizational structures of an enterprise. And the integration of distributed application engineering with multimedia and cooperation support (in both facets, workflow and CSCW) is hardly approached at all.

To summarize, we see a heavy need for a new era in software technology, enabling effective production of

- large and distributed
- media-integrated
- cooperative (workflow-type and CSCW-type)

application software. We want to use the term **CMA** (cooperative media-integrated applications) in the remainder to denote this type of application. In the following, we will recall major lessions which we learned in predecessor projects and describe the embedding of our work in the current project context. We will then sketch major aspects of an adequate framework for the development of CMAs.

2 Background

2.1 Experiences and Requirements

For the proposed framework, we draw from the experience gained in several predecessor projects, undertaken jointly with other universities and industrial research groups. Three of these projects are to be mentioned here.

Project Docase [GZH90,MGH93] lead to a modeling / programming framework for

large (non-multimedia) distributed applications. The key lessons learned were as follows:

- Distributed object-oriented programming languages form an excellent basis for the development of large distributed applications. In particular, they exhibit three features which we summarize as *distribution transparency*:
  - such languages provide for location independent operation ('method') invocation (i.e., local and remote 'procedure' calls are semantically equivalent).
  - since design and implementation usually yield a large number of *small* objects, decisions about the distribution of these objects over a target network can be deferred to installation time; in particular, distribution aspects do not have to be taken into account during design.
  - if object migration is supported, distribution decisions can even be altered at runtime.
- Adequate modeling and design support is crucial for the successful development of large object-oriented programs: a huge number of objects have to be handled on the implementation level, making object-oriented programs even less managable than conventional ones. Docase supports modeling on the base of socalled "object categories", a concept that will be explained and expanded in the context of the new framework proposed in this paper.
- For mission-critical software, lots of so-called "operational" aspects need attention, such as authorization, authenication, reliability, and accounting. With current software engineeering techniques, all these aspects are mangled into the mainstream application code. Therefore, a new modularization concept was developed, called 'program superimposition', extending work described in [Kat93]. In chapter 3, we will motivate the application of this concept in the context of the proposed new framework.

In the Nestor project [MüS92], we developed services and tools for the development of cooperative media-integrated *courseware* (computer-based teaching or instruction material). On one hand, Nestor lead to a series of different multimedia and CSCW tools, such as a software video codec, a tool for recording Xwindow output to a file for further processing, and a tool which augments Xwindow applications for cooperative use. On the other hand, applying these tools in computer aided instruction made us aware of the need for a much more integrative, customizable approach to courseware (and software) construction. In this respect, we learned the following lessons from Nestor:

- the marriage of object-oriented concepts and hypertext leads to an appealing and powerful new concept
- support for *distributed* multimedia aspects is essential; distribution *transparency* is particularly important, but needs substantial additions to the known approaches mentioned above (i.e., distributed object-oriented techniques)
- while applications can be tranparently augmented for a primitive level of cooperation support, real "cooperation awareness" needs a substantial development effort; at present, *generic* CSCW programming tools hardly exist.

The third predecessor project, DIRECT, represents a 'hand-crafted' sample CMA which served for gathering requirements about the framework proposed in chapter 3. DIRECT supports physically distributed research or engineering teams with a set of integrated cooperation and task management tools. It is centered around the "issue-based design" metaphor for organizing ill-structured problems, originally described in [Pot89]. Direct showed that

- an integrated CMA can support the users much better than a set of isolated tools
- such a CMA can be partly made up from pre-built components, provided the components are highly customizable (programmable)
- workflows and human cooperation, as supported by a CMA, are highly interrelated; a common modeling concept for both is desirable.

2.2 Project Organization

The work described in chapter 3 is embedded into the B.I.G. embracing project whose major sponsor is Digital Equipment:

B stands for Berkom, a stretegic project in Germany, which runs under the auspices of the research and development spin-off of the german PTT, DTBerkom Several multimedia services and tool suites are built as part of the project. Digital is one of the major industrial partners in the project, our group is invoved in the development of multimedia collaboration (MMC) and multimedia mail (MMM) tool suites [ADH93].

I stands for Items, the project in which the CMA development framework is developed by our group. This part is described in more detail in the following section.

G stands for 'Gigaswitch', a new switching technology developed by Digital with an aggregate throughput of several Gigabits per second. Gigaswitch connects FDDI-based computers, but in contrast to standard FDDI, a star topology connects the FDDI stations point-to-point, so that the full FDDI speed can be exploited by every station. due to the use of dedicated point-to-point links, Gigaswitch networks can assure quality of service parameters that make them much more suitable for multimedia communication than standard FDDI. Future members of the Gigaswitch familiy are supposed to assure transition to ATM.

In summary, the B and G parts of the BIG project allow our Items activities to be embedded into a large testbed which relates our work to many national and international activities, and which provides dedicated and multimedia-proof high-speed connectivity; Berkom, in addition, provides tools and software pieces which we can use to evaluate our integration concepts.

3 Proposed Framework

3.1 Overview, Item-Oriented Programming

This chapter is devoted to the description of a framework for the development of CMAs. The project and development environment are called 'Items', referring to the idea of 'item-oriented programming'. This term is meant to show that on one hand, we use object-oriented programming as our basis, but on the other hand, we extend the functionality of objects for better support of cooperation and multimedia aspects (the term item can be understood as the acronym for '*i*conic *te*lecooperating *m*ultimedia object)

Since we want to give an idea about the overall Items system, some details will have to be related to more in-depth papers mentioned in the references. Some parts of items represent revised and extended versions of modules and concepts developed in the predecessor projects mentioned. The Items system is still under construction since the project is in a relatively early stage. In particular, the system still consists of loosely coupled parts whose integration has not been achieved yet. The experiences gained in predecessor projects, however, makes us confident that most of the serious problems have been resolved and that the concepts described in the remainder represent a feasible approach.

We will first give a coarse view of the overall Items architecture and the major tool-building-tool which we are using (section 3.1). In section 3.3, we will concentrate on the item-oriented design and programming model. Thereby, we will give a short overview of the top-level so-called 'item categories' available. In sections 3.4 through 3.7, we will describe the functionality of some of these categories in more detail, concentrating on the following aspects for brevity: an introduction to our support for distributed object-oriented programming in general (very coarsely discussed), support for distributed multimedia, support for workflow-type and CSCW-type cooperation, and support for re-usable hypertext structures.

3.2 Items System Architecture

The Items system represents a whole software engineering environment with several modeling, programming and monitoring tools, code management support, and integration facilities on the data, tool, and UI layers. Fig. 3.1 concentrates on those components of the Items system which are crucial for the design and execution of item-oriented CMAs.

The most important tool is called *VIP* (visual item-oriented program design tool). VIP accompanies the programmers throughout the design and implementation phases. It integrates both the graphical and the textual programming metaphor. VIP supports graphical programming of the coarse and fine grained design, including dynamic aspects. Code is automatically generated from design. There is no virtual boundary between design and implementation; instead, textual programming of software modules can be carried out in the context of the graphical entities managed by VIP. Thus, a seamless integration of design and programming, of visual and textual program development is supported. The sections below will describe the elements of graphical programming ('item categories').

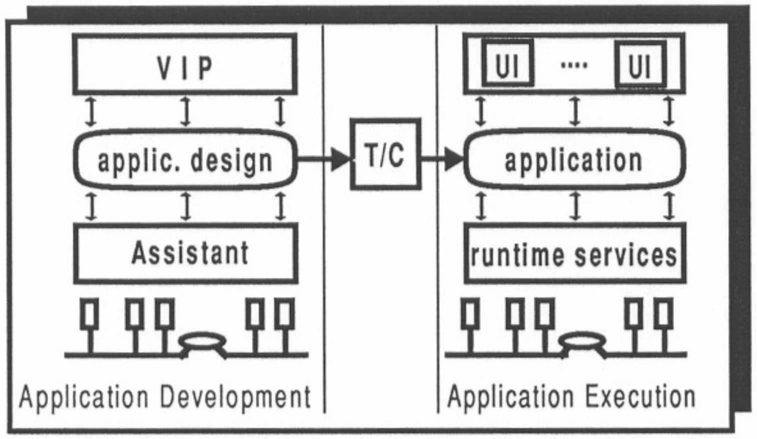

Fig. 3.1. Core tools and services of the Items system

VIP is built using a very generic tool-biulding-tool developed in our group, called ODE. ODE (object-oriented design editor) was used to build many different graphical tools and interfaces in the past. It is based on a very flexible object model that can be customized to the needs of the tool under construction. For customization of ODE, a Lisp dialect is used; C and C++ routines can be linked into the code. Different graph layout algorithms are attached to ODE via a special layout subsystem called LAMA. Currently, ODE is extended for use in distributed software engineering environments and for cooperation with heterougeneous programming tools.

Apart from VIP; the so-called *Items Design Assistant* has to be mentioned. The development of this assistant stems from the fact that many design notations (graphical, textual-language-based, or else) are associated with a design *method,* i.e with strategies and rules to follow during design. Most design *tools,* however, hardly support the respective method; this is especially true for methods which are not totally 'formal', because their respective methods cannot be expressed in the context of the design notation. The Items Design Assistant uses a separate formal language to describe an informally given development method, mapping it to building blocks and operations of the design methods, to artifacts of the application under development, and -most important - to different types of artifacts which represent the reasoning about the design (steps, issues, arguments, etc.). The concept of a Design Assistant has been successfully used in the Docase project already (cf. [MGH93]).

As fig. 3.1 indicates, an application is transformed into executable code by special transformation / compilation tools. Thereby, the graphical information is transformed into source code and interleaved with the textual program parts. This leads to purely textual source code which is compiled. The current version of Items supports a distributed version of C++ (called DC++) which was developed at our Institute. We intend to support a more elegant distributed object-oriented programming in the future, a successor of the DOWL language developed in our group [Ach91]. DOWL is a strongly typed multiple-inheritance language which supports object migration even for objects which are in execution. The language currently supported, DC++, is not as elegant (due to the ancestors, C and C++), but powerful enough to reach distribution transparency in the context of Items.

At runtime, an item-oriented CMA relies on a number of sophisticated runtime services, partly described below. In particular, we will describe the service which provides distribution transparency in the presence of multimedia objects. Fig. 3.1 also indicates the decoupling of the core of item-oriented applications from their user interface part, which is described further below, too.

3.3 Item Categories

Rationale: So-called *class libraries* have become a common approach in the object-oriented community. In analogy to 'function libraries' of conventional languages, such class libraries contain the definitions of pre-built 'kinds of objects' (called 'classes' or 'types', depending on the programming language) which together represent the core functionality of a certain problem domain. E.g., 'Interviews' is a very well-known C++ class library for the programming of window-based graphical interfaces, making live much easier than with plain Xwindow programming. Two major problems restrict the value of class libraries:

- When an application programmer uses a class library, he will usually have difficulties to learn the syntax and semantics of the pre-defined object classes. He will mostly have to depend on manuals and on the level of checking a compiler can provide; the compiler, however, will 'understand' little of the semantics of the object classes and nothing about the semantics of the way in which they are to be used in an application - most checking will be on the syntactic level.
- Large applications, CMAs in particular, will exploit many different facets and aspects of functionality. A programmer will therefore want to use several different class libraries. However, it is a common experience in large software projects that only a very small number of different libraries or APIs can be integrated with a single application program, due to the necessary effort to understand the different 'mind sets', 'models', and ways of usage associated with pre-built libraries or APIs.

In the items project, we try to overcome these problems by using the so-called 'category approach'. Categories are pre-built object classes in the first place, but their semantics are 'understood' by the programming tools and they cover the complete software development, not only a small domain like graphical interface programming.

For further discussion, we refer to fig. 3.2. The somewhat 'condensed' layout in this figure shows a tree with a root (parent node, called 'item') and two levels of child nodes (first level: shapes only, no icons; second level: shapes *and* icons); at one point, the third level is included as well (child nodes of 'interactor').

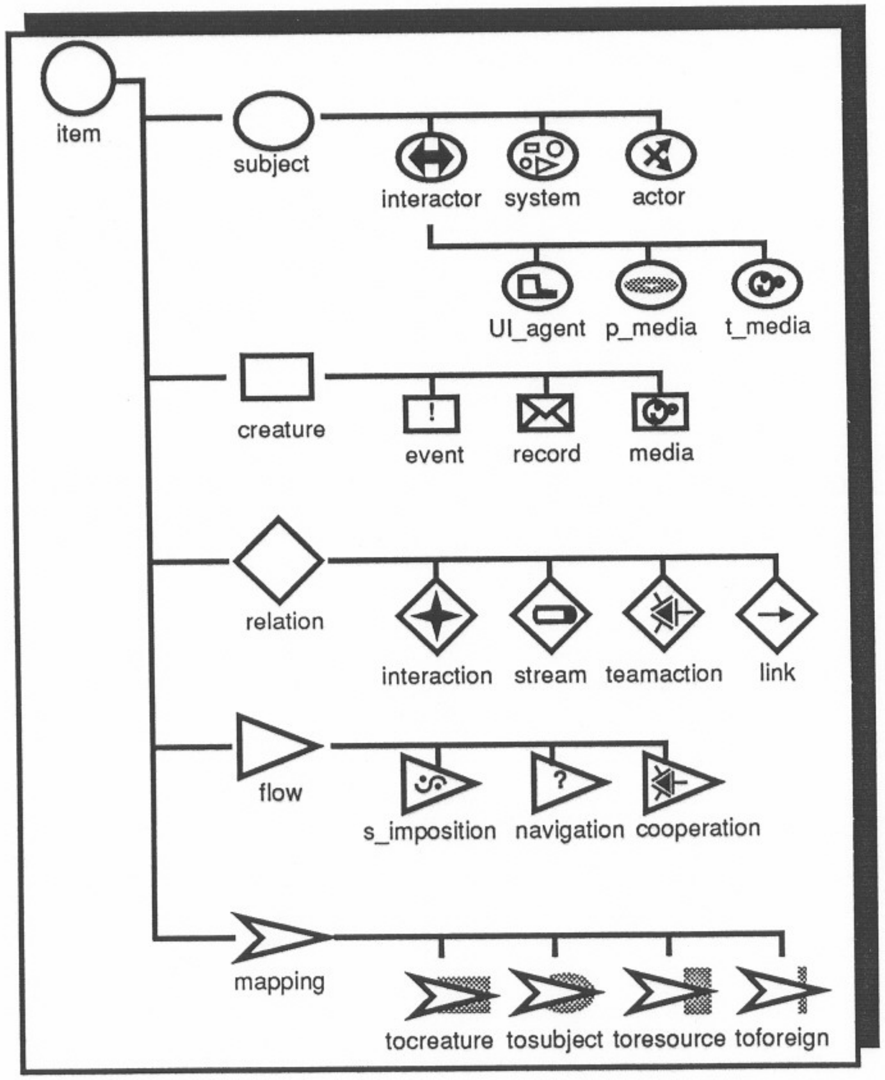

Fig. 3-2 Item Categories

The main functionality of item categories can now be explained more precisely; thereby, the differences to the common 'class libraries' should become obvious:

- A whole item-oriented CMA is made up of 'items': every class definition used in an application must be derived from one (or several) of the categories shown in fig. 3.2. Of course, every object (or item, to be more precise) in an application must in turn be an *instance* of such a class definition.
- The first level of categories below root (subject, creature, relation, flow, and mapping) represents 'orthogonal' 'meta' categories. 'Orthogonal' means that any class definition must be derived from *exactly one* of these five top-level categories (the items in an application can thus be divided into five disjunct sets). 'Meta' means that instances cannot be directly created from one of these categories, but only from their children and further descendants (sub-classes).
- The VIP graphical/textual program development tool offers sophisticated support to the software engineers as they develop an item-oriented CMA. Starting from the palette of item meta-categories, the tool allows to select an actual category of choice, to create or modify subtypes of this category (class-based view), to define instances of a class or to redefine the class from which they are derived (instance view), or to reason about various dynamic relationships between classes or instances (scenario view). Since VIP 'knows' the semantics of the categories, it guides the user as he specifies parts, relationships and behaviour of a class and as he interconnects items in the design.
- The Items Assistant can be used to arrange the VIP-based micro-level design operations (adding or modification of classes or instances) into macro-level steps according to an embracing design method and general design rules.

In the remainder of this section, we will give an overview of the item categories, to the level of detail depicted in fig. 3.2 (note that further sub-categories exist):

Subject. The subject meta-category represents the entities that make up the backbone 'configuration' of an application. An initial configuration of subjects must be defined within any application; subjects can be added, removed, or migrated between nodes only by means of explicit 'configuration changes'. In contrast to 'creatures', subjects typically have threads of control of their own, i.e. they can act asynchronously and in parallel to other items. The following major sub-categories of 'subject' exist:

- *interactor:* this category provides interfaces to human-perceivable information representations; such information is supposed to come from or go to parts of the system which are external to the applications, such as multimedia devices or archives, and multimedia user interfaces. Three sub-categories of interactors are depicted in fig. 3.2:
  - *UI_agent:* this category plays an emportant role for ensuring high portablity of item-oriented applications. It is motivated by the experiences with window-based graphical user interfaces (GUIs) based on common windowing systems such as Motif or Windows. These systems usually offer a very low level interface between the core application and the GUI: the 'look and feel' has to be described in terms of geometric data, and many window-system

specific details have to be programmed (he GUI server code itself cannot be changed usually). This problem lead to the development of several systems which support the use of several GUIs transparently. These approaches encapsulate differences between different windowing systems, making applications more portable.
The UI_agent category in items goes one step further: it encapsulates differences between UIs that are based on different media and metaphors (MUIs). UI_agents come with a skeletal, high-level "application-to-UI protocol" (where, e.g. the window-based concept "menu" is replaced by the more abstract concept "selection", "scope-change" replaces "zoom/pan", etc.) and with implementations for different MUI types (first version: window-based and speech-enahnced). Both the protocol and the MUIs can be customized to accomodate specific application needs and further MUI types.

- *p_media:* 'permanent' multimedia archives are encapsulated by this category, covering the range from simple file store to document storage servers, multimedia kiosks, and multimedia databases. The interface to a p_media item may include queries and discrete and continuous media storage/ retrieval. The 'subject' nature of p_media allows it to model, e.g., asynchonous ('overnight') delivery of information.
- *t_media:* the 'transient' t_media items encapsulate capturing and presentation devices such as cameras, microphones, displays, scanners, etc. Asynchronous behaviour, here, may occur from the possibility for the user to switch the device on or off or to carry out other local operations which the computer can become aware of. Special synchronization support for multple media is given according to the concept described in [BHL92].

- *system:* the system category compensates one of the lacks of conventional object-oriented systems: while such systems provide an excellent means for modelling 'kind of' relations by inheritence, they support 'part of' relations only marginally, usually by providing an 'aggregate' concept. 'System' items represent the major concept for hierarchical (and overlapping) decomposition of an application configuration. They enhance the 'part of' concept considerably: e.g., a system can offer an interface to the external items which completely hides the internal structure and contents. The role-resolution concept (described further below) can be used for mapping the 'external' interface to the content items of a system. Based on its own thread, a system controls its internal configuration (number and types of configurable items, their interconnection via 'relation' items, etc.) as well as the role-resolution.
- *actors*: very similar to the 'active object' concept available in some object-oriented systems, actors may contain several (light-weigth) threads of their own for which they manage concurrency and synchronization; actors dispatch incoming requests autonomously.

Creature. The term 'creature' indicates that items of this category are created at runtime. They are not considered part of the so-called 'application configuration'.

Migration of 'creatures' can be determined autonomously by the 'object placement service' of the runtime system (unless explicitly forbidden by the programmer). Intuitively spoken, creatures model parts of an application which are created and destroyed 'in masses', such as mails, route slips which accompany parts in a production plant, or database records. Three sub-categories are relevant:

- *event:* this category models the classical 'software trap' concept and is used for asynchronous activation of subjects.
- *record:* on one hand, records represent the 'default' modeling for 'standard' objects which do not differ from the kinds of objects found in a conventional object-oriented system. On the other hand, the system support for migration, propagation and routing of records makes their 'journey' through a distributed system more efficient. Like media (see below), records are exchanged between subjects, either directly or via 'relations'.
- *media:* the media category provides the unique system model for all kinds of single media and multimedia data. Its interface includes a set of generic operations (display, create, edit, etc.) and a 'quality of service' concept. More details will be given in section 3.4.

Relation. With the 'system' category explained above, we compensated for deficiencies in the object-oriented 'part of' relation. The relation category now compensates for deficiencies in the object-oriented 'knows' relation. in object-oriented systems, one object 'knows' another one when it knows its unique identifier (typically, this identifier is stored in a so-called 'instance variable'). Any object may call operations of any other object it 'knows', thereby establishing a temporary communication relation. Thus, the communication relations between objects are 'hidden' in the application program and neither explicitly modeled nor easily tracked at runtime. For items, we introduce an explicit 'relation' category which can be used to model binary and n-ary 'communication paths'. The following four sub-categories add more functionality to this concept.

- *interaction:* this category provides so-called communication schedules as described in [ScG91]. They support multiparty relations and multi-stage communiction (several subsequent interrelated communications). The communicating subject items involved are described via roles (which are resolved at runtime).
- *stream:* in order to transport continuous media, 'streams' can be defined. Such streams make efficient use of the underlying transport system and do not provide direct access to the data transferred (these data have to be extracted via the interface of the 'media' category described above). Streams are also handled by the distributed runtime service described in 3.4
- *teamaction:* the concept for generic CSCW-type cooperation support (described in section 3.5 below) is founded on the idea that standard objects (more precisely: specific item categories such as media and record) can be extended to team-objects by adding a 'teamaction' context to their operation. See below for more details.

- *link:* this relation category adds basic hypertext functionality to the items model. Links are mainly used in the context of 'navigations' (see below and section 3.6). A link connects subjects cas its 'sources' and 'destinations'; the latter can be determined at instanciation time (static link) or at the time a 'navigation' traverses the link (computed link).

Flow. In contrast to relations and similar to subjects, flows contain threads of their own. A flow is a kind of 'global thread' which may traverse many different subjects; it is, however, long-lived and persistent (a flow may take on for months) and not bound to any particular subject. A flow is not 'hosted' in a particular network node and traverses node boundaries at will.

- *s_imposition:* as mentioned in chapter 2, the extended superimposition concept used in items represents a way of coping with the multitude of 'operational aspects' (accounting, security, reliability...) of large CMAs. The s_imposition category contains both a 'global thread' (in the context of which certain operational aspects have to be considered) *and* a description about how the operational aspects are to be intertwined with the 'global thread' (and with all the items called from there. This describtion uses parametrized code fragments and so-called 'filters' which determine the details of how the fragments have to be intertwined with other pieces of code (items, parts therein, parameter settings etc.), based on a kind of formal semantics. We will not describe this fundamental concept in more detail since it is not central to the problems of cooperation and multimedia; rather, we point to a fundamental article about program superimposition and to our own work in this respect [Kat93, Heu90].
- *cooperation:* this common modelling concept for both workflow-type and CSCW-type cooperation will be described in 3.5
- *navigation:* a navigation can be seen as a 'global thread' which carries a 'user' through a series of subjects and links. Apart from the classical understanding of 'hypertext navigation' which supports users in reading complex documents, navigation items may support individuals to carry out multiple activities in a coordinated way, e.g., during a software development process. As such, navigations complement the 'multi-user' cooperations with a 'single user' global thread. The navigation concept will be elaborated in 3.6.

Mapping. Much of the runtime flexibility of item-oriented CMAs is due to the four role-resolution concepts included, called mappings:

- *tocreature:* this mapping can be used if the determination of the specific creature used in an application involves a non-trivial selection at runtime. E.g., different creatures may play a certain creature_role at different times; in this case, the embracing program defines a 'creature_role', and the 'tocreate' mapping defines both the required properties and the process of determining the specific creatures to fulfil the role.
- *tosubject:* in analogy to the above, 'tosubject' defines how 'subject roles' are resolved. This category will usually be much more important than the above one, used in relations and flows in many ways.

- *toresource:* resources in items comprise humans, devices, nodes, and schedulable resources such as bandwidth and compute time (corresponding sub-categories exist). As to humans, the 'toresource' maps person roles (e.g., as used in cooperations) to individuals at runtime). Network nodes and schedulable resources are taken into account by the multimedia object service described in 3.5, by the 'object placement' service mentioned in 3.4, and by 'interactor' items.
- *toexternal:* CMAs will hardly ever be built from scratch. Apart from item-oriented software modules to be reused, 'legacy software' will have to be included. In order to connect such legacy software to an item-oriented CMA, we use a concept which conforms to the CORBA 'object request broker' standard issued by the 'OMG' standardization body (object management group).

3.4 Distributed Object-Oriented Programming

Since this paper concentrates on the cooperation and multimedia aspects of CMA programming, we want to mention the general aspects of support for large distributed programs in items only very briefly. During the above section, however, it should have become evident that the item-oriented programming concept covers all aspects of distributed programming, not just cooperation and multimedia in particular. This reflects our central goal to support *media-integrated* applications. In addition to the distributed object-oriented concept, which already provides an excellent ground for the development of large distributed programs, items provides particular support for substantially enhanced modeling and target customization.

Modeling is enhanced through the category concept which allows software engineers to structure their application design in a 'standardized' way; category-based designs can be better understood and more easily communicated to peer software engineers, managers and users. Relation categories and flows help to make the 'behaviour' of the system much more evident, superimpositions provide for modularization and re-use of 'operational' aspects, and systems support better decomposition.

Target customization means that the same application code can be used in different or evolving runtime environments. For this end, too, items provides serveral concepts: sophisticated role resolution is supported with the mapping categories; automatic dynamic object placement is carried out by a runtime service which automatically migrates creatures (and selected other items) according to a heuristics-based algorithm (which minimizes inter-node communication, cf. [Sch90]); superimposition, again, is important since it allows to superimpose different operational algorithms at different times or for different target environments.

3.5 Distributed Multimedia Object Service

As discussed earlier, distribution transparency is essential for development frameworks for large distributed programs in general. Thus, it has to be assured for item-oriented programming in particular. However, distribution transparency is particularly hard to maintain in the presence of multimedia information. e.g., fig. 3.3 depicts part of an application scenario which works with a multimedia 'stream' established between a p_media source and a two t_media sinks. The operation semantics behind this scenario might be "play video x on the displays of persons y and z". Distribution transparency, in this context, means that the programmer need not care about the location of the video and the displays, neither about the available network bandwidths, storage formats, supported compression formats etc.

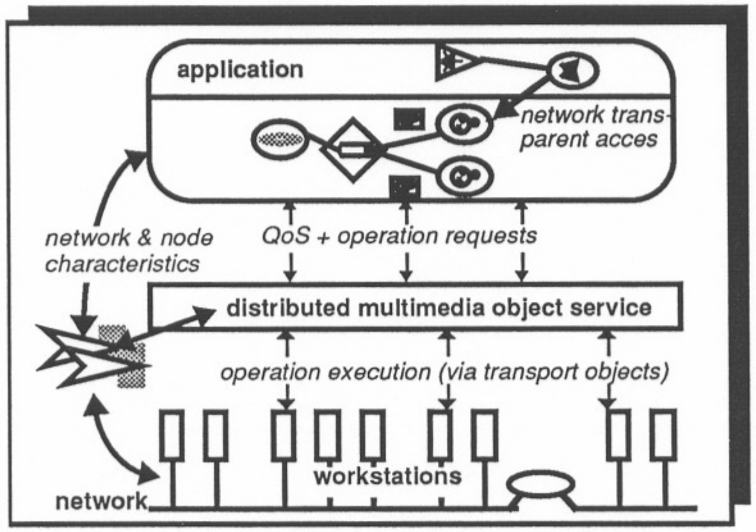
application
network trans-parent acces
network & node characteristics
QoS + operation requests
distributed multimedia object service
operation execution (via transport objects)
workstations
network

Fig. 3.3: Distribution Transparency for Multimedia Objects

To this end, we developed a distributed multimedia object service called MODE [Bla92]. Mode is based on descriptions of the 'environment' of an application (networks and workstation characteristics), given as 'toresource' mappings. It uses the item-oriented multimedia object model and therefore 'understands' the quality of service requirements and the operation calls issued by an application.

MODE is based on internal models of 'presentation objects' (cf. t_media and p_media items), 'information objects' (cf. media items), and 'transport objects' (optimized for transport in a network, invisible to the item-oriented program). In our example, the 'environment information' and the source and sink 'presentation' objects are used for an optimization at runtime.

Based on a 'decision tree', this optimization yields a so-called 'path' at runtime, composed of (source/sink) presentation objects and of transport objects (in a different example, 'information objects' could be included as well). This path represents the sequence of media transformations, transports, and manipulations which alltogether carry out the required operation under the given QoS constraints. In the example, MODE might find out that there is a low-bspeed bridge between

source and destination requiring video compression, and that next to the users' workstations there is a station which can carry out decompression in hardware.

3.6 Cooperation Support

The cooperation support is centered around the item categories 'teamaction' and 'cooperation', based on predecessor work described in [Rüd91].

The *cooperation* category describes 'global threads' of parallel and sequential activities with a mix of flow-oriented and rule-based techniques. A partial, flow-oriented set of activities can be described as a 'task'. Such a task is described as a set of parallel and sequential activities, each of which is guarded by activation conditions and by required results which mark successful completion. Tasks can be assembled into cooperations based on a set of ordering rules (this leads to a high degree of flexibility in the ordering of tasks).

Tasks refer to person_roles and 'teamaction' items. Teamactions augment operations of creatures or subjects for use in a cooperation context. This way, any item-oriented program can be easily extended for 'cooperation awareness', i.e. for use in a CSCW-type and/or workflow-type cooperation context.

Fig. 3.4 indicates how methods can be augmented for cooperative use. Standard items are shown to the left, subject_roles to the right. The important part is the teamaction context in the middle colomn which determines the choice among different pre-defined alternatives (about synchronism, visibility, and mode) about the cooperative use of the operations.

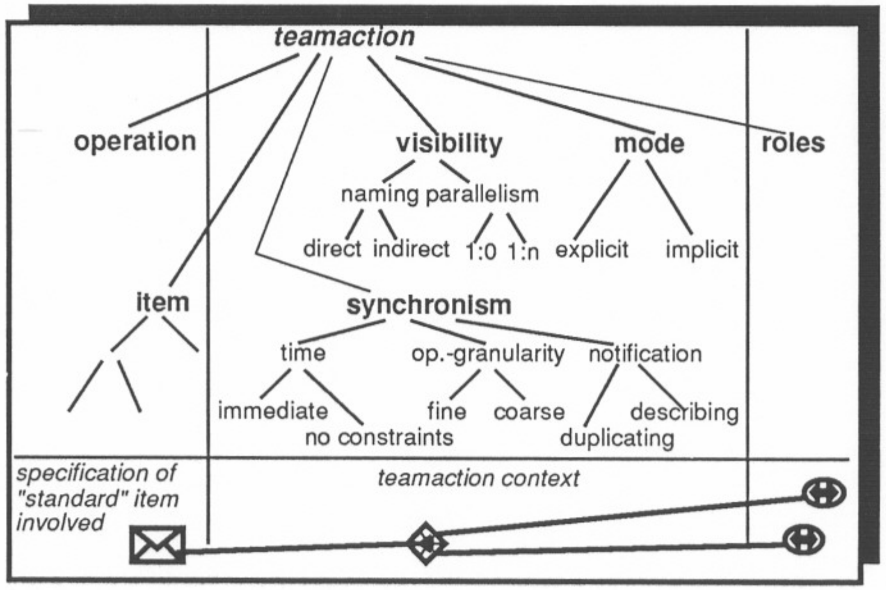

Fig. 3.4: Teamaction context of an item

[Rüd91] explains how these choices cover all kinds of CSCW-interactions available in all CSCW applications known from the literature. The more in-depth discussion in this reference brings evidence to the advantages of the items-oriented cooperation approach over other ones known from the literature. But even from the brief discussion in this section, it should become clear that our approach is neither limited to a specific subset of possible cooperative applications nor concentrates on the CSCW aspects of software engineering alone (such as the [HBP93]), and that it is superiour to class library approaches such as GroupKit [RoG92].

Through the use of subject_roles, a teamaction can refer to an activity without determining whether it is carried out manually or automatic. This way, a workflow can remain unchanged while individual activities are automated in an enterprise.

3.7 Hypertext support

The item categories 'link' and 'navigation' form the basis for hypertext support in the Items system. A serious drawback of existing hypertext systems lies in a lack of typing support. While every hypertext can be interpreted as a graph consisting of 'nodes' and 'links', not all hypertext systems include a sophisticated (e.g., object-oriented) typing concept for such nodes and links; virtually *no* hypertext system supports typing for the hypertext networks (graphs) made up from nodes and links.

Typing support for nodes and links is relatively easy to achieve. In items (where pre-defined types are categories), every sub-category of the subject category can be handled as a 'node type'; every link type must be a sub-category of the relation category and must have the 'link' category as one of its super-categories.

Typing support for hypertext networks, however, is more difficult. The central question is: 'what *is* a network type' (in other words, a 'class' or 'family' of hypertext networks)? In items, we use an approach which was first discussed in the context of a predecessor project [Müh91]. A type of a hypertext network is thereby described as a navigation category, based on a visual graph grammar depiction as indicated in fig. 3.5.

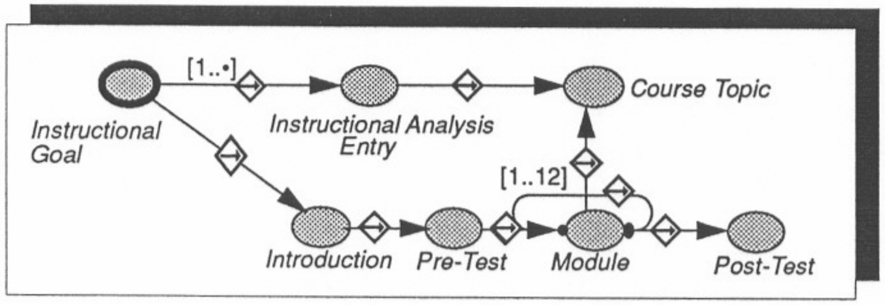

Fig. 3.5. Example hypertext network type

In fig. 3.5, the 'construction' part of a network type (i.e., navigation category) definition is shown. This part describes the rules which determine the graph structure of any hypertext of the respective category. The figure illustrates an example taken from instructional software. The link in the upper left of the figure is marked with an interval [1..°]; this means that at least 1 link of the given type has to originat from the subject 'instructional goal', but there may be as many such links (with the same source node) as desired. In the lower right of the picture, a loop with interval [1..12] is drawn. This shows that in a hypertext of the given category, as many as 12 nodes of type 'Module' may exist in a row, interconnected by the link type indicated in the loop. Apart from the construction rules mentioned, further ones exist of course, as described in [Müh91].

Apart from 'construction' part as just described, a navagation category consists of a 'navigation' part which describs rules and constraints that determine the way in which an individual thread traverses a network of the category described. The experiences with predecessor projects of items have shown that by programming navigation rules in the context of a hypertext category instead of re-programming it for every instance, considerable gains in re-usability and understandability of hypertext-based programs can be made.

4 Summary

We presented an overview of a proposed modeling / programming framework for cooperative media-integrated applications, called Items. The proposal is based on the experience gained with two predecessor frameworks and a sample CMA. It is embedded into a suite of several high-speed networking projects which provide the testbed for our work.

The work described is not completed, but the major conceptual elements have been tested in smaller individual prototype versions of the framework.

References

[Ach91] Achauer, B.:
Distribution in Trellis/DOWL.
Proc. TOOLS 5, Santa Barbara, USA, July 1991, pp. 49 - 59
[ADH93] Altenhofen, M, Dittrich, J., Hammerschmidt, R., et al.:
The BERKOM Multimedia Collaboration Service
Proc. ACM Multimedia '93, 1.-6.8.1993, Anaheim, CA
[Bla92] Blakowski, G.:
High Level Services for Distrib. Multimedia Applications
Based on Application Media and Environment Descriptions.
Proc. ACSC-15, Hobart, Australia, January 1992.
Australian COmputer Science Communications, 14(1), 1992, pp. 93-109

[BHL92] Blakowski, B., Hübel, J., Langrehr, U., Mühlhäuser, M.:
Tool Support for the Synchronization and Presentation of Distributed Multimedia
Butterworth Jl. on Computer Communications, December 1992. pp. 611 - 618
[GZH90] Gerteis, W., Zeidler, Ch., Heuser, L., Mühlhäuser, M.:
DOCASE: A Development Environment & Design Language f. Distrib. O-O
Applications. Proc. TOOLS Pacific '90, Sydney, Australia, Nov. 1990, pp. 298 - 312
[HBP93] Hill, R., Brinck, T., Patterson, J., et al.:
The Rendezvous Language and Architecture
CACM 36 (1), Jan. 1993, pp. 62-67
[Heu90] Heuser, L.:
Processes in Distributed Object-Oriented Applications.
Proc. Tool'90, Karlsruhe, Germany, Nov. 1990, pp. 281 - 290
[Kat93] Katz, S.:
A Superimposition Control Construct for Distributed Systems
ACM ToPLaS 15 (2), April 1993, pp. 337 - 356
[MGH93] Mühlhäuser, M., Gerteis, W., Heuser, L.:
DOCASE - A Methodic Approach to Distributed Object-Oriented Programming
to appear in CACM 36 (10), Sept. 1993
[Müh91] Mühlhäuser, M.:
Hypermedia and Navigation as a Basis for Authoring / Learning Environments
AACE Jl. of Educational Multimedia and Hypermedia, Vol 1, No. 1, 1991, pp. 51 - 64
[MüS92] Mühlhäuser, M., Schaper, J.:
Project Nestor: New Approaches to Cooperative Multimedia Authoring / Learning
in: I. Tomek (Ed.): Computer Assisted Learning, Springer Verlag, Berlin etc. 1992,
pp. 453 - 465
[Pot89] Potts, C.:
Recording the Reasons for Design Decisions.
Proc. IEEE 11th Int. Conf. on SW Engineering, Singapore, May 1989, pp. 418 - 427
[RoG92] Roseman, M., Greenberg, S.:
GroupKit: A Groupware Toolkit for Building Real-Time Conference Applications
Proc. CSCW '92.
[Rüd91] Rüdebusch, T.:
Development and Runtime Support for Collaborative Applications.
in: H.J. Bullinger: Human Aspects in Computing. Elsevier Science Publishers
Amsterdam 1991, pp. 1128 - 1132
[Sch90] Schill, A.:
Mobility Control in Distributed Object-Oriented Applications.
Proc. IEEE Intl. Conf. on Computers and Communications,
Phoenix, Az, March 1989, pp. 395-401.
[ScG90] Schill, A.., Gerteis, W.:
Communication Schedules: An N-Party Communication Abstraction Mechanism for
Distrib. Applications. Proc. 10th ICCC '90 (Nov. 1990, New Delhi, India), pp. 643-651.