Issues in Multimedia Software Development

Max Mühlhäuser
University of Linz, Altenbergerstr. 69, A-4040 Linz, Austria
max@tk.uni-linz.ac.at

Abstract

Multimedia has become a topic of high interest, and R&D contributions to multimedia fill piles of CD-ROMs, journals and books. Out in the field, however, only a comparatively small set of out-of-the-box niche multimedia applications is used; developing customized multimedia applications, tuned to the needs of a particular organization, remains much of a myth, not to speak of integrated enterprises in which full-size multimedia information and communication is used throughout. The MMSD workshop has been organized as an attempt to help filling the gap between the availability of multimedia solutions and their customized, integrated use in the field. This introductory article positions MMSD in the range of multimedia R&D domains, recalls the categories of contributions relevant to software development in general ('solution space'), describes particular characteristics of multimedia software ('problem space') together with corresponding sample research contributions, and positions the contributions to the workshop in these spaces.

1. Multimedia and Software Development

Many have tried to define the term multimedia. We do not want to add yet another definition here, but draw the reader's attention to the three most relevant multimedia aspects in the context of multimedia software development (MMSD):

- *conceptually,* multimedia is marked by the fact that computers are used to *coordinate* and process *multiple media,* some of *which are time-dependent*
- *technically,* a considerable number of the challenges are associated with *digitized audiovisual information*
- *practically* spoken, *distributed multimedia applications* (DMMA) represent probably the largest market potential, larger than that of multimedia PCs. DMMA also represent the vision of an upcoming merge of telecommunications, consumer electronics, and computer science.

The enormous R&D efforts in multimedia computing concern almost every domain of computer science. In an attempt to categorize these efforts, one may find the following domains of multimedia research:

1. *physical representation* – e.g., formats and *compression techniques* for digitized audiovisual information; ways to represent computer-generated media (e.g. MIDI, VRML)
2. *perception* – topics like handwriting recognition, audio world-spotting, video face recognition and the like
3. *system support* – appropriate real-time scheduling in operating systems, performance guarantees in communication protocols, integration of perception techniques (see above) into multimedia databases, etc.
4. *logical representation* – models / architectures of media (such as object-oriented class hierarchies), multimedia (e.g., as to synchroinzation schedules), multimedia applications (such as videoconference configurations), etc.;
5. *development support* – authoring tools for multimedia presentations and hypermedia documents, APIs and frameworks, etc.

Of course, any development support is related to a corresponding logical representation of what is developed, so that the last two topics above must both be considered in the MMSD context. Physical and logical representation of multimedia have been separated above, despite their blurred borderline (see, e.g., the MIDI example), because the former is mainly considered in the context of perception and system support while the latter is mainly related to software development.

While the above-mentioned five-tiered classification scheme may help to distinguish concerns in multimedia as a whole, it is helpful in our context to categorize multimedia software development in further detail. In order to do so, we want to adopt a classification which is, in different variants, commonly used in the context of software quality assurance. There, five areas of constructive means for software quality assurance are distinguished (as opposed to analytic means like debugging and process metrics), cf. fig. 1. We want to adopt these five means as the 'solution space' for multimedia software development, i.e. as the areas in which approaches and solutions are seeked.

1. **principles:** the underlying 'groundrules', ranging from simple concepts like separation of concerns to entire programming paradigms such as object-orientation
2. **models:** the 'languages' in which software can be expressed, ranging from reference architectures to calculi
3. **methods:** the procedural approaches available, from algorithms to all kinds of 'plans', based on the principles, and used by or applied to the models and tools.
4. **tools:** the instruments at hand, using and supporting the

0-8186-7511-6/96 $5.00 © 1996 IEEE

models and methods

5. **processes:** encompassing, typically phase-oriented ordering of decisions and steps and their relation to the other four elements.

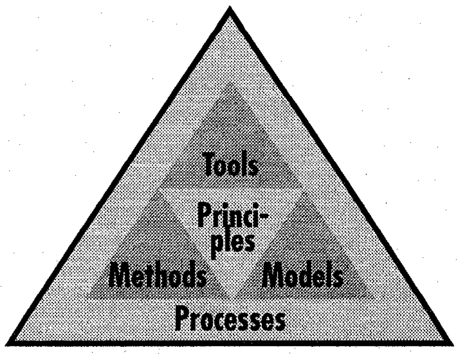

Figure 1. The 'solution space'

2. Problems and Sample Solutions

Research about multimedia software development support can be classified based on the 'solution space' above, but also based on the 'problem space', i.e. by the particular characteristics of multimedia software which have to be reflected when seeking solutions.

Table 1: The' problem space'

characteristic	meaning (simplified)
media	complex evolving media types replace trad. data types; impact: devices, formats, conversions, cplx. operations
time	notion of time is part of many media types, relevant for composition & processing
synchron-ization	desired ordering & relation of media in time & space must be specified and realized; distribution / exception problem
QoS	parameters & characteristics of media are constraint by statistical values; different sets at different abstraction levels; particular relevance for DMMA
streams	singular message transmissions can or should not be regarded at most abstraction levels; sets of time-related message exchanges are grouped into streams
semantics	configuration & 'meaning' of media, MM documents, MM applications and appl. scenarios (e.g., workflows) to be expressed & processed

We will use a somewhat simplistic scheme (cf. table 1) of six categories to describe the major characteristics of multimedia software: media, time, synchronization, quality-of-service (QoS), streams, and semantics. As this introductory article is mainly supposed to help the reader in navigating through the remainder of the MMSD'96 proceedings, we will not provide a survey of the state of the art for these six categories. For better understanding, we will however cite a few advanced contributions to these categories. While it is obvious that the six categories are not entirely disjunct (e.g., synchronization is based on the notion of time), the detailed discussion below will hopefully substantiate the choice made.

Media: proper handling of media is a prerequisite for multimedia. Problems stem, e.g., from the sheer size, the fact that new media, formats, operations etc. must be accommodated, from the perception problem, and from the lack of integration with hardware and software engineering (in contrast to traditional data types). The object-oriented approach seems to be ideal for introducing an open number of new media types, together with the operations defined. Media class hierarchies promise to encapsulate formats, (e.g., compression) algorithms, standards, etc. Actually building and implementing a media class hierarchy requires second thoughts, however. The discriminators for designing the class tree are not obvious (e.g., at which level should time-dependent media be distinguished from time-independent ones, at which level digitized AV media from 'model-based' computer-generated media?"); the reflection of issues like formats and conversions and the smooth integration of devices as objects must be accomplished. Class interfaces can hardly be defined such that all future extensions can be anticipated (cf. upcoming possibilities like the extraction of a 3D model of a person from a conventional digital video). The class hierarchy as proposed in the IMA (Intl. Multimedia Association) standard [3] illustrates some of the effects of the considerations discussed: an actual object (cf. AVXwindowdevice) is a rather odd mixture of multiple inheritances and aggregations, formats and devices are more central to programming than the actual media.

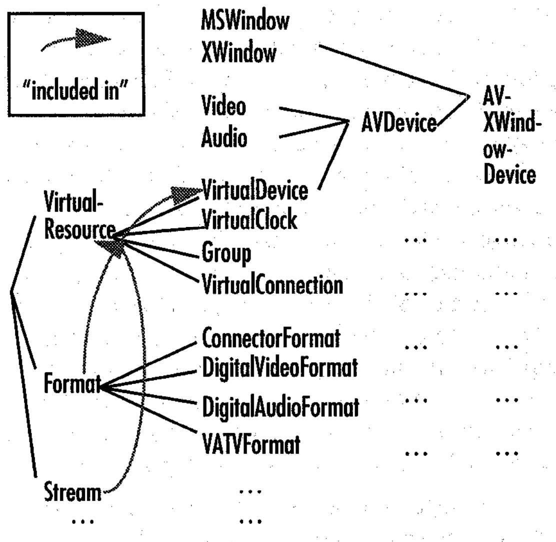

Figure 2. IMA class hierarchy, excerpt

Finally, an object model as discussed is oriented towards technology instead of 'real-world objects' as it should: e.g., a conversation among teleconference participants must be *designed* to use particular media instead of deferring this choice to runtime.

Time: Continuous media impose particular real-time constraints on active objects like those which capture, compress, or present audio or video. They are particular as, e.g., in the multimedia context, *not* meeting a deadline is not a necessarily a failure (e.g., video frames which are not ready for presentation 'in time' may be presented with slight delay or dropped; both "exceptions" may not even be visible to the user). Most known approaches in this area are based on (thread-, process- or system-wide) *clocks*; usually, the overall real-time behavior of such systems cannot be predicted (as to, e.g., the missed/met deadline ratio).

A different approach, *synchronous programming;* usually requires programs that are written as finite state machines in a special 'predictable' programming language. For these reactive programs, maximum state transition times can be computed, and in consequence, the overall system response time can be computed and compared to a given mean event inter-arrival time (remember that for continuous media, the number of events per time unit is given in advance). Thus, synchronous programming allows to check in advance whether or not the components of a program will meet their deadlines. At the Lancaster University Distributed Multimedia Laboratory, *reactive objects* (objects defined using the synchronous programming concepts) are investigated as a complement to active objects (=objects with threads) [8]. In the multimedia context, reactive objects might be used to guarantee timely response to events. E.g., a reactive object might control a chain of active objects (producers, transformers, consumers) and decide, based on the state of the active objects, whether they should continue to process a multimedia data sample or drop it. In addition to method interfaces, the object model for active/reactive objects must comprise states, constraints, and events. The approach proposed makes it possible to define a border line between guaranteed and non-guaranteed multimedia performance (reactive and active objects). The value of synchronous programming is not unanimously appreciated, however, in part because of the need for specialized languages and systems, in part since it does not yet provide much of a clue to the overall 'suitability' of a software configuration (e.g., as to the above-mentioned missed/met deadline ratio). Nevertheless, reactive objects currently represent probably the most advanced approach to coping with time in media.

Synchronization: Multimedia synchronization concerns both an inter-object aspects ("smooth delivery of samples") and an intra-object aspect (parallel and serial play-out schemes for multiple media 'tracks'). Synchronization also concerns both time (as just indicated) and space (arrangement of visual media on a common presentation device such as a monitor), yet research has concentrated a lot on the time-related problems. In terms of the solution space, both adequate models and appropriate methods are required. As to the latter, the above-mentioned approaches to handling time and the below-mentioned QoS approaches can be combined and extended so that inter- and intra-object synchronization can be achieved. We will therefore concentrate on the modeling / specification issue.

Off-the-shelf multimedia authoring tools often use a common time axis as the means for specifying duration and relative ordering of media tracks. Problems such as failure to include elements with unpredictable duration lead to extensions like virtual axes or the reference point model used in MODE [1], see below. Many other approaches exist, such as scripts (powerful but difficult to handle by non-programmers), hierarchical synchronization trees and event tables (both do not scale up very well), finite coordinate systems (cf. HyTime[7]; they cover spatial synchronization have been criticized wrt. unpredictable duration and intuitiveness), and PetriNet-related approaches such as OPCN [4] (the formal background enables automatic inspection/verification, but threatens intuitiveness).

As to intuitive usability, the time axis and related models are considered superior since they are based on the simple left-to-right parallel/serial ordering of media tracks. The above-mentioned MODE approach does not only combine left-to-right ordering with support for non-predictable duration elements, it increases intuitiveness by using time axes only at places in the synchronization specification where no continuous media are presented – and thus, no inherent timing is given – and where no non-predicable timing applies (e.g., in a part of a multimedia presentation where several slides are shown, but no audio or video).

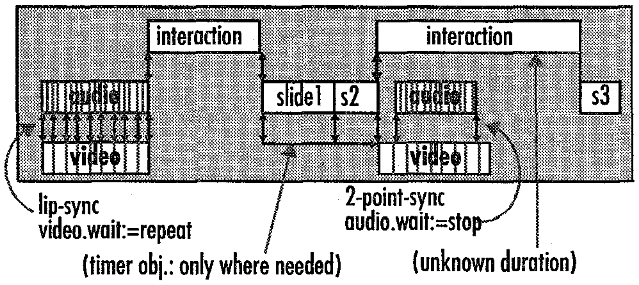

Figure 3. MODE sample synchronization schedule

MODE also provides concepts for defining a) intra-object synchronization; b) the *level* of synchrony between parallel tracks (consider, e.g., a video track with lip-synchronous audio vs. background music audio); and c) the exception handling if synchronization deadlines are not met. The third aspect mentioned is particularly important in todays distributed systems which cannot (yet) guarantee QoS requirements throughout the network. In addition, MODE keeps object identity and coherence – in many other approaches, certain synchronization constellations may require the fragmentation of logically coherent media objects into 'meaningless' pieces.

Quality-of-Service (QoS): Generally spoken, specifying values for QoS parameters can be interpreted as *"putting constraints on parameters"*. QoS is particularly relevant to multimedia since there, the amount of data processed can be varied considerably as a means for trading off user-level quality/fidelity against processing 'cost' such as transmission and processing time, public communication cost, etc. QoS concerns all software levels as an orthogonal aspect. User-level QoS, for instance, concerns aspects of human perception which can hardly be quantified (recent publications have proposed a "QoS-by-example" approach to accommodate this fact [10]). At the media interface which is best understood in terms of QoS, important QoS parameters reflect communications issues (end-to-end delay, bit and packet error rate, etc.), issues of the overall processing chain (jitter i.e.relative distortion of continuous-media samples etc.), and issues of the individual media (such as, for video, frame rate and resolution, speed ratio etc.). Other important QoS values can be expressed by combinations of the above-mentioned ones.

QoS consideration is particularly difficult for a number of reasons: QoS parameters represent random variables, thus *different statistical values* may be relevant (mean, min./max., standard deviation, burst lengths, etc.); QoS is supposed to be *negotiated* between users, applications, and different service-providing components in an iterative process which comprises the difficult mapping between different sets of QoS parameters (see above); QoS can usually *not be fully guaranteed* today due to a lack of net/ OS and hardware support.

The straightforward approach of just adding QoS-related parameters to the parameter lists of APIs or to method calls of foundation classes does not reflect any of the above-listed problems. As an example, one may look at the above-mentioned IMA standard [3] which provides only rudimentary solutions to the problems mentioned: only three QoS parameters (delay, jitter, bandwidth) are considered; the statistical nature can only be reflected by indicating an accepted bandwidth (min. and max. values); the co-existence of QoS-supporting and QoS-unaware middleware can be reflected by choosing among three guarantee levels.

Iterative QoS negotiation only makes sense in the context of explicit durable connections between media-processing objects, not for 'single-shot' communication paradigms like remote procedure call and method call. This increases the demand for streams as discussed below.

Application-transparent QoS optimization for DMMA is still an academic topic today. E.g., MODE [1] provides distribution-transparent object-oriented programming and runtime support. It uses logical names in method calls such as, say, video.play(student.workstation,QoS-list). MODE retrieves the location of relevant media objects and presentation devices from network-wide resource files. Further information about the media involved is taken from the application specific media classes (inherited from system-defined meta-classes). Based on this information, a planning process is triggered which plans the processing chain from information sources to sinks via transformation steps such as compression and comprises all necessary QoS negotiation between the components involved.

Streams: the sections above raised the need for explicit connections between media processing units. The simplest form of multimedia processing chain, consisting of a sender, transmission channel, and receiver, is often denoted as stream. In the object-oriented context, transmission of samples over a stream becomes automated instead of method call initiated. Rather, method calls are introduced for stream management (setup, disconnection, ...) and manipulation (pause, ffd, ...).

We will refer to the IMA approach again as an example. IMA has defined an elaborate stream model; in the model, the interconnected processing elements consist of an aggregate object with virtual devices, formats and ports; the interconnection is achieved via the object class 'virtual connection'; several other objects are involved in a stream definition. As fig. 4 is supposed illustrate, the IMA approach requires the configuration of quite a number of predefined objects in order to obtain a single point-to-point stream – this concept appears to be rather overloaded.

Rather than providing a complicated point-to-point stream model, object-oriented approaches may reflect high-level stream issues in the future such as multihop, multicast topologies, along with system-support for assembling these. System level configurations (network connections) and the logical (application process) level might be distinguished (here, configuration semantics comes into play, see below).

It should be retained that the stream issue strongly suggests a link concept as part of the underlying principles; an explicit link concept is neither present in the remote procedure call paradigm nor in the object-oriented paradigm (remember that IMA models streams as objects, not as links between objects). Conceptually spoken, the hypermedia paradigm may turn out to be more 'natural' for modeling processing elements and streams (as hypermedia nodes and links) – approaches for merging object-oriented and hypermedia concepts may therefore be particularly interesting here in the future.

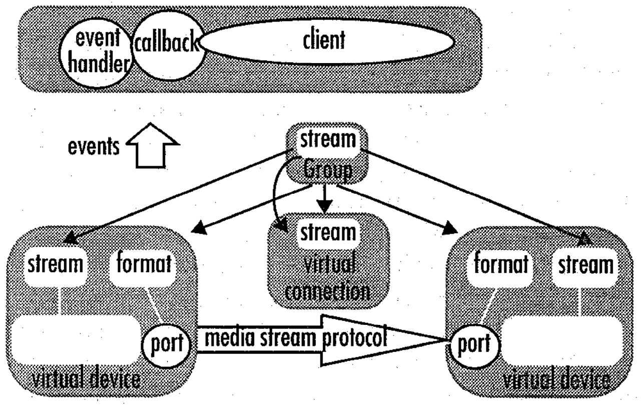

Figure 4. IMA stream example

Semantics: as table 1 indicates, the term semantics here denotes the issue of expressing the meaning, structure, relations, and configuration of media, multimedia, and applications. Apart from the problem of perceiving digitized audiovisual information (cf. item 2, 'perception', in the first section), the problem of expressing multimedia semantics is strongly related to the model aspect (cf. fig. 1).

E.g., synchronization models as discussed above represent one kind of model for expressing multimedia semantics. Hypermedia models concentrate on the relation between multimedia information entities. Other approaches support the modeling of the use of multimedia in the context of an application configuration / architecture, such as the Items approach [6], cf. fig. 5 where a setup for electronic meeting assistance is graphically modeled: the 'vcr' node indicates that conversation between meeting participants is recorded to disk. The meeting assistance software (denoted EMA in the figure) keeps a list of topics and controls the order and timing of their being addressed.

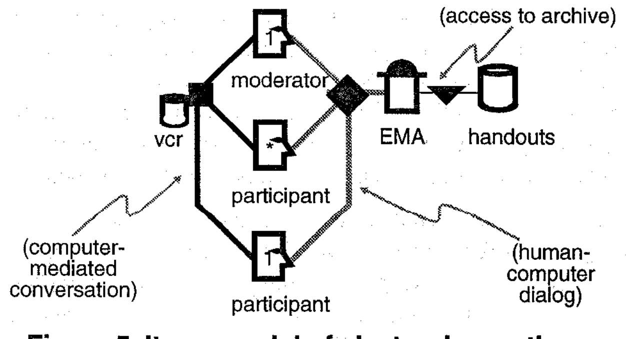

Figure 5. Items model of electronic meetings

Multimedia application models (often called 'architectures') are of central importance for the advancement of MMSD for two major reasons. One reason is that based on a common understanding about a model, building blocks can be furnished (as a basis for component-based programming). Given the complexity of DMMA software, this possibility is absolutely crucial for the advancement of the field. The second reason is the possible advancement of media perception, indexing, and handling in general that may become possible; this issue shall be explained with the example from fig. 5. There, the rich semantic context of the application model (topic & timing 'knowledge' of the EMA component) can be used, e.g., in order to produce an automatic index of the audiovisual material stored at the 'vcr' archive (AV fragments indexed by topics/timing) for easy retrieval of audio and video.

Apart from Items, frameworks such HeiMAT [9], DAVE [5], and Simon Gibbs' framework [2] have also concentrated on multimedia application models ('architectures'). In Items, further modeling support exists in order to express the interrelation between different configurations (e.g., in the workflow sense).

3. MMSD'96 contributions

The above discussion allows us to classify the MMSD contributions as to the problem and solutions spaces which they address.

As discussed, the modeling of (distributed) multimedia applications is a key issue, both for the predominant requirement of supporting component-based programming (essential for bridging the gap between the availability of technology and its customized and integrated use in the field, i.e. for fulfilling the key initial requirement stated) and for gaining more 'knowledge' about media contents.

Therefore, it is not astonishing that a majority of *seven MMSD papers* are related to this issue. Among these seven, Blum & Molva [11] concentrate on layered structuring of the overall software, as a base for standardization and API construction; their major categories addressed are therefore 'model' and (directly related) 'semantics'. Qian et al. [12] follow the same goals, but present a fairly different approach with an emphasis on standards (CORBA) compliance. Bochmann et al. [13] put an additional focus on the QoS problem domain, while Fritzsche [14], Frick [16], and Barth [17] all emphasize the provision of building blocks; they may therefore be associated with the 'method' and 'tool' solution space, too. Eliassen & Nicol [15] are different from the other six 'architecture' papers as they put a strong emphasis on the 'stream' problem domain and provide 'methods' in addition to models.

Three papers clearly concentrate on the 'process' solution domain: Fedchak et al. [18] look at electronic publishing, thus on authoring more than on traditional programming, just like Lux [19] and Morris&Finkelstein [20]. Thus all three belong to the multimedia document subdomain of the semantics problem domain.

Another group of three papers concentrates on the synchronization problem domain, providing approaches which belong entirely to the 'model' solution category. Vuong et al. [21] extend the OPCN notation mentioned in chapter 2, Vazirgiannis et al. [22] cover synchronization in time *and* space, an aspect which had been reported as underrepresented above, and Shish et al. [23] adopt the Z notation and thus start a new 'line' of synchronization approaches in addition to the ones listed.

The two papers on hypermedia obviously address the 'semantics' problem domain. Rösch & Bäntsch [24] review two standards and thus remain in the 'model' domain which relates directly to 'semantics', while the authoring support discussed by Leidig & Rösch [25] is clearly associated with the 'tools' class.

While models for application semantics (architectures) and models for multimedia / relation semantics (synchronization, hypermedia) have been covered, none of the above has emphasized models for the semantics of particular media. This subdomain of the 'model' solution domain is covered by Koumpis et al [26] – who add user modeling and interaction modality to the conventional text media type – and by Li et al. [27] – who add security to video.

Since underlying principles are of course addressed in almost all the papers implicitly or explicitly, the list of papers can be said to reflect the entire solution space. As to the problem space, only the time aspect is not covered, so that altogether the list of contributions to the MMSD workshop represents an excellent representation of the topics of interest.

4. Acknowledgments

The excellent coverage of the relevant areas of MMSD deserves special thanks to all authors. In particular, however, the co-chairs are indebted to the program committee for their support. We are thankful that a representative selection of world leading experts in MMSD have devoted their time and energy to the success of the first International Workshop on Multimedia Software Development. Last but not least, the organizers of ISEW and the editor-in-chief of IEEE CS press, Bob Werner, shall be assured of our particular gratitude.

5. References

[1] B. Blakowski, J. Hübel, U. Langrehr, and M. Mühlhäuser, "Tool Support for the Synchronization and Presentation of Distributed Multimedia," *Butterworth Jl. on Computer Communications*, Vol. 15, No. 10, 1992, pp. 611-618.

[2] S.J. Gibbs and D.C. Tsichritzis, *Multimedia Programming - Objects, Environments& Frameworks*, Addison-Wesley 1994.

[3] Interactive Multimedia Association, "Multimedia Systems Services," Draft Rec. Practice, 1995.

[4] T.D. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects" *IEEE Jrnl. Selected Areas in Communications*, Vol. 8, No. 3, April 1990.

[5] R. Mines, J. Friesen, and C. Yang, "DAVE: A Plug and Play Model for Distributed Multimedia Application Development," *Proc. 2nd ACM Intl. Conf. Multimedia*, Oct. 15-20, 1994, San Francisco, ACM New York, pp. 59-66.

[6] M. Mühlhäuser, "Modeling and Design of Complex Cooperative Software," *Proc. 1st IEEE Conf. Engineering of Complex Systems*, Ft. Lauderdale, FL, Nov. 6-10, 1995.

[7] S. Newcomb, N. Kipp, and V. Newcomb, "the HyTime Hypermedia/Time-based Document Structuring Language," *CACM*, Vol. 34, No. 11, Nov. 1991, pp. 67-83.

[8] M. Papathomas, G.S. Blair, G. Coulson, P. Robin, *Addressing the Real-Time Synchronization Requirements of Multimedia in an Object-Oriented framework*, IS&T/SPIE Proc. on Multimedia Computing and Networking, Vol. 2417, 1995.

[9] R. Steinmetz and J. Fritzsche, "Abstractions for Continuous-Media Programming," *Computer Communications*, Vol. 15, No. 6, July 1992, pp. 396-402.

[10] A. Vogel, B. Kerhervé, et al. "Quality of Service Management", *IEEE Jl. Multimedia Sys.*, Vol. 2, No. 2, Summer 1996.

Articles published in these proceedings:

[11] Ch. Blum and R. Molva, "A SW Platform for Distributed Multimedia Applications".

[12] T.Qian,S.-M. Tan, and R. Campbell, "An Integrated Architecture for Open Distributed MM Computing".

[13] G. v. Bochmann and B. Kerhervé, "Architectural Design of Adaptive Distributed Multimedia Systems".

[14] J. C. Fritzsche, "Multimedia Building Blocks for Distributed Applications".

[15] F. Eliassen and J. R. Nicol, "A Flexible Type Checking Model for Stream Interface Binding".

[16] O. Frick, "Multimedia Conferencing Systems as Building Blocks for Complex Cooperative Applications".

[17] I. Barth, "Configuring Distributed Multimedia Applications using CINEMA".

[18] E. Fedchak and L. Duvall, "An Engineering Approach to Electronic Publishing".

[19] G. Lux, " A Design Methodolgy to Maintain Consistency between Functional Behaviour and Multimedia Presentation".

[20] S. J. Morris and A. C. W. Finkelstein, "Integrating design and development in the production of multimedia documents".

[21] S. Vuong, K. Cooper, and M. Ito, "Specification of Synchronization Requirements for Distributed Multimedia Systems".

[22] M. Vazirgiannis, Y. Theodoridis, and T. Sellis, "Spatio - Temporal Composition in Multimedia Applications".

[23] T. Shish, D.-A. Chiang, and H.-Ch. Keh, "Formal Specification of Multimedia Authoring".

[24] P. Rösch and M. Bäntsch, "Reviewing two Multimedia Presentation (quasi-) Standards".

[25] T. Leidig and P. Rösch, " Authoring MHEG Presentations with GLASS-Studio".

[26] A. Koumpis, Ch. Karagiannidis, and C. Stephanidis, "Adaptations of the Text Media Type: Addressing the User's Perspective".

[27] Y. Li, Z. Chen, S.-M. Tan, and R. H. Campbell, "Security Enhanced MPEG Player".