RWTH Aachen University
Media Computing Group
Prof. Dr. Jan Borchers

Post-Desktop User Interfaces
WS 2006/2007

Multimedia Architecture

Ines Färber (251268)
and
Alexander G. M. Hoffmann (229544)

18.01.2007

Tutor: Thorsten Karrer

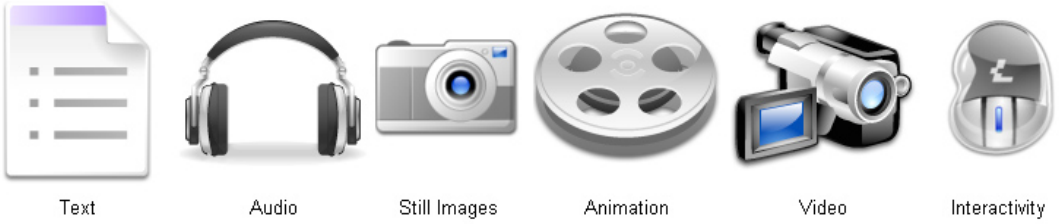
Abstract

The area of multimedia applications is a field in which cross disciplinary cooperation plays an increasing role. Besides frictionless collaboration and presentation issues of the different media they also have to satisfy the hard real-time and interaction requests of the end-users. This paper presents some interesting approaches, which are supposed to solve problems that occure while creating multimedia software.

Contents

1 Introduction to Multimedia

The first challenge of this paper is the definition of multimedia architecture, for which no clear containment exists. Multimedia is an integration of multiple of the following digital media into one big media: text, audio, still images, animation, video and interaction. In some cases a combination of these media is only regarded as multimedia, if at least one of these is time-dependent. Figure: [12]



Typical multimedia applications these days are video games, video conferencing systems, speech recognition systems and animation presentations. Since the architectures presented in this paper can be used for some of these applications, we will get back later to them. A multimedia application deals with the creation, manipulation, presentation, storage or exchange of multimedia-based data. Especially for manipulation and presentation, interactivity, multi-tasking and parallelism play an important role. To allow the user an untarnished multimedia experience during interaction these applications have to cope with hard optimisation constraints such as real-time performance, low latency and precise synchronisation.

To combine more than one medium a good synchronisation and a good collaborative work are required; all influences have to be well-regulated. An architecture should identify which processes have influence on the calculation at what time. Since every combination of digital media leads to diverse requirements we present different architectural approaches to solve typical multimedia application problems. One aspect considered in all presented architectures is the one of interactivity with the end-user. Furthermore, we provide case studies to introduce examples of multimedia applications of which the presented architectures form the basis.

2 SAI - Software Architecture for Immersipresence

One Multimedia-Software-Architecture is SAI. Francois defines SAI as follows:

> SAI (Software Architecture for Immersipresence) is a new software architecture model for designing, analyzing and implementing applications performing distributed, asynchronous parallel processing of generic data streams. [7]

SAI is a designing tool that gives a general formalism for a multimedia architecture. This approach tries to analyse new and existing systems easy and gives a construct for system

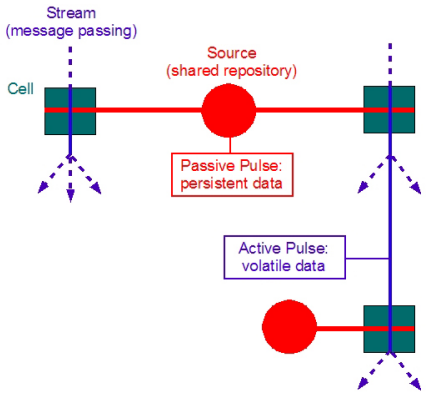


Figure 2.1: Overview of SAI elements [7].

implementation. This architecture combines a high bandwidth with a low latency. It offers a distributed implementation of algorithms and their integration into complex real-time systems.

The problem about actual complex systems is the integration of, e.g. cross-disciplinary algorithms. Resource intensive activity causes high latency and unforeseen problems are hard to locate. SAI makes an easy integration of underlying libraries, native code, or algorithms possible and provides qualities like efficiency, scalability, extensibility, reusability, and interoperability.

Dataflow architectures are simple and intuitive, support parallel and distributed processing. The limitation of efficiency and modelling power, just as problems with shared data excess cause that dataflow architectures are a non-optimal solution. Blackboards are not adapted to online and real-time processing. SAI combines all these missing features.

The fundamental principles of SAI are the explicit account of time both in data and processing models through processing centres (cells) and sources, the distinction between persistent and volatile data, and the asynchronous parallelism, which supports the properties of lower delays.

The architectural style of SAI is explained in a graphical way in figure 2.1.

Cells are represented as squares, sources as circles. Source-cell connections are drawn as fat lines, while cell-cell connections are drawn as thin arrows crossing over the cells. When color is available, cells are colored in green (reserved for processing); sources, source-cell connections, passive pulses are in colored in red (persistent information); streams and active pulses are colored in blue (volatile information). The graphic-based notation technique is realized in the SAI application: VisualSAI.

SAI is divided in conceptual and logical levels. The conceptual level includes the set of cells and sources, the inter-connections, and the description of the different tasks. The logical level covers passive and active filters inside the processing centres and the structure of passive, persistent pulses that enter and leave the sources.

2.1 Case Studies

SAI has been used for the design and implementation of various experimental systems.

2.1.1 MuSA.RT

MuSA.RT Music on the Spiral Array. Real-Time [4], is a system for real-time analysis and interactive visualisation of tonal patterns in music. MIDI input is processed, analysed and mapped in real-time to the Spiral Array, a 3D model for tonality, revealing tonal structures such as pitches, chords and keys [7]. The active triad is indicated by a coloured triangle. Red indicates major, blue minor. The grey dots outside the spiral array are the actual pressed keys and the green lines visualise the played main melody. The VisualSAI Conceptual graph of MuSA.RT is given in the following figure 2.2.

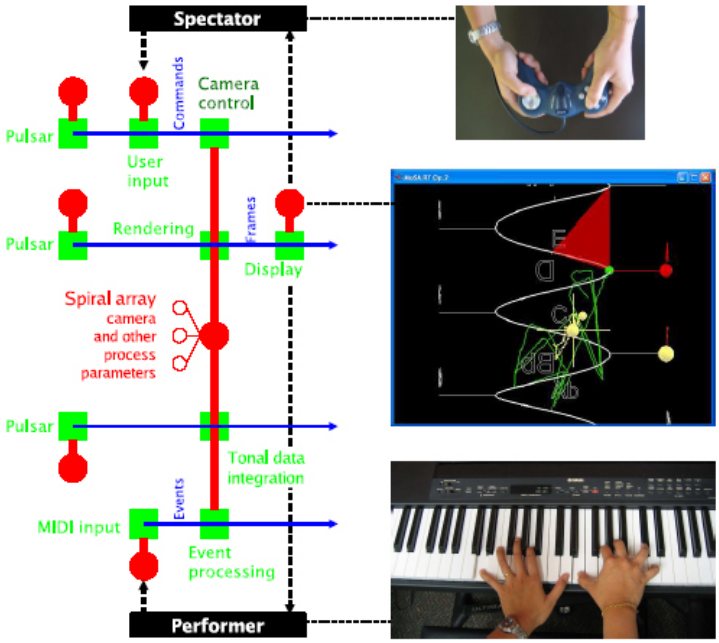


Figure 2.2: Application flow graph for MuSA.RT [7].

2.2 Conclusion

The goal of SAI is to provide a universal framework for the distributed implementation of algorithms and their easy integration into complex systems, that provides qualities such as efficiency, scalability, extensibility, reusability and interoperability [8].

New are the fundamental principles. The modularity of the style facilitates the division of code development, testing, and reuse, as well as fast system design and integration,

maintenance and evolution. A graph-based notation allows intuitive system representation at the conceptual and logical levels, while at the same time mapping closely to processes.

The SAI style achieves optimal system latency and throughput, and is a framework for consistent representation, and efficient implementation of complex systems.

3 Fran - Functional Reactive Animation

Programming of profoundly interactive multimedia animations including sound, pictures, video, 2D or 3D graphics has long been a complex task for specialists only, mainly because a clear differentiation between modelling and presentation was missing. Programmers therefore had to manage the questions "what is an animation" and "how to present it" at the same time and single-handed.

Animation programmers must explicitly solve the problem of conceptually continuous and parallel manipulation of parameters in animation presentations through low-level display libraries running on a sequential computer.

Fran (Functional Reactive Animation [5]) affords a declarative (what to do) instead of an imperative (how to do) kind of programming. The author has complete freedom to define an interactive animation without caring about details of discrete, sequential presentation, which are relinquished to the underlying implementation.

Fran is a high level vocabulary and provides a collection of data types and functions, adapted to programming of animations. Fran is domain-specific embedded in the declarative language Haskell. Haskell is best applicable for a declarative approach of programming modelled animations because of its valuable properties like non-strict semantics, higher-order functions, strong polymorphic typing, systematic overloading and many more [9].

The key-concepts in Fran are its notions of behaviours and events. Behaviours are time-varying, reactive values, while events are sets of arbitrarily complex conditions, carrying possibly rich information [5]. This concept involves a modelling approach established by four features:

1. Temporal modelling: The core of this modell are behaviours, which are time-dependent values. These first-class values [11] are built up compositionally and the programmer is able to express concurrency almost naturally and implicitly.
2. Event modelling: Events are first-class values as well. Either they refer to happenings in the real world (e.g. left mouse button pressed) or to conditions based on animation parameters (e.g. collision). Different events can be combined and so reach an arbitrary degree of complexity. The programmer can factor this complex logic into semantically rich, modular building blocks.
3. Declarative reactivity: Considerations here focus on behaviours, that are reactions to events. Semantics of these reactive behaviours are still the same as for non-reactive ones.
4. Polymorphic media: As the variety of time-varying media fits into a common framework of behaviours and reactivity many operations provided by Fran are polymorphic

and can be applied to all types of time-varying values. However all this media and their parameters have their own type specific operations.

The collection of recursive data types, functions and primitive graphic routines, which helps realise this concept of modelling, is provided with formal denotational semantics [5], including a proper treatment of real-time. The semantic domains will be presented in Chapter 3.1.

The specific feature of Fran is its implicit treatment of time. Because events can be specified as Boolean functions of continuous time and can become true for arbitrarily brief time periods and instantaneously it is challenging to generally detect them. In Fran this problem is solved by using a method for event detection based on interval analysis [10]. Interval analysis concentrates on predicate events, which are events depending on conditions of behaviour parameters. Sole Sampling of behaviours is not enough, because behaviours need not to evolve gradually. Instead a conservative interval bound is produced. Bounds are defined by the values that are allocated to the behaviour over a given time-interval I. Using this interval bound one can determine whether the event takes place or not.

3.1 Formal Semantic domains of Fran

Behaviours and events are treated as a pair of mutually recursive polymorphic data types on which one can define operations. The abstract domain of time, called *Time* is defined as $time = \mathbb{R} + \mathbb{R}$. Elements of the second version of $\mathbb{R}$ are prefixed with $\leq$, for example $_{\leq}42$ means "at least 42". The bottom element is defined as $\perp_{Time}$. Together with the following ordering on *Time* this domain is a complete partial order:

$$\begin{array}{ll} x \sqsubseteq x, & \forall x \in \mathbb{R} \\ {}_{\leq}x \sqsubseteq y \ if \ x \leq y, & \forall x, y \in \mathbb{R} \\ {}_{\leq}x \sqsubseteq_{\leq} y \ if \ x \leq y, & \forall x, y \in \mathbb{R} \end{array}$$

Via this chain-like definition of time (in particular every sub-chain has a least upper bound) values of *Time* include partial elements, so that in some cases we know that time is "at least" some value, even if we do not know exactly what the final value will be. Elements of *Time* are most useful for approximating the time in which an event occurs.

The abstract domains of polymorphic behaviours (α - behaviours) and polymorphic events (α - events) are identified as $Behaviour\alpha$ and $Event\alpha$. α - behaviours are interpreted as functions from time to α - values, producing the value of a behaviour b at a time t : **at** : $Behaviour_{\alpha} \rightarrow Time \rightarrow \alpha$.

α - events are interpreted as simply non-strict pairs $Time \times \alpha$, describing the time and information associated with an occurrence of the event: **occ** : $Event_{\alpha} \rightarrow Time \times \alpha$.

3.2 Conclusion

Fran is a system for animation programmers, which allows to concentrate on issues of modelling, leaving presentation details to the underlying implementation. This one step towards simplification of the animation programming process is done by the replacement of imperative techniques with these declarative ones. The current implementation of Fran runs under Hugs, a Haskell implementation and the authors C. Elliott and P. Hudak expect marked performance improvement when Fran is running under the Glasgow Haskell Compiler (GHC).

Possibilities for improvement may be more features for 2D, sound and 3D. Additionally, future work lies in improving performance through the use of standard compilation methods as well as domain-specific optimisation techniques.

A related work is TBAG [6] a system that uses a continuous time model and has a syntactic flavour similar to Fran's, however reactivity is handled imperatively. The ideas underlying Fran also formed the basis of Microsoft's *DirectAnimation*, a COM-based programming interface, accessible through conventional programming languages like Java, Visual Basic, JavaScript, VBScript and C++.

4 Multimedia Information Services Enabling

The paper of "Multimedia Information Services Enabling: An Architectural Approach" [3] deals with extendable and scaleable multimedia information management systems. The system should be easily adjusted to new data types and should not lose performance when it has a growing workload.

4.1 An Architecture for Multimedia Service Enabling

The core functional needs of multimedia retrieval services are supported through the following modules.

The **Raw data server module** contains the possibility to handle "raw" multimedia data. To describe the raw material, metadata is used. To handle this kind of data the **metadata module** that offers querying and modelling functions is used. To present query results and interact with the user, another module is required; The **user interfaces and presentation module** supports different interaction and presentation functions. The **user profiles and personalisation module** which has been added because of the increasing importance of user profiling and personalisation in multimedia applications. The **communication and data transfer module** is connected to all other modules. Any kind of communication is passing through this module. It supports implementation independent support for transportation needs. Transport functions covered are remote method invocation, event channels, BLOB transfer, and streaming functionality [3].

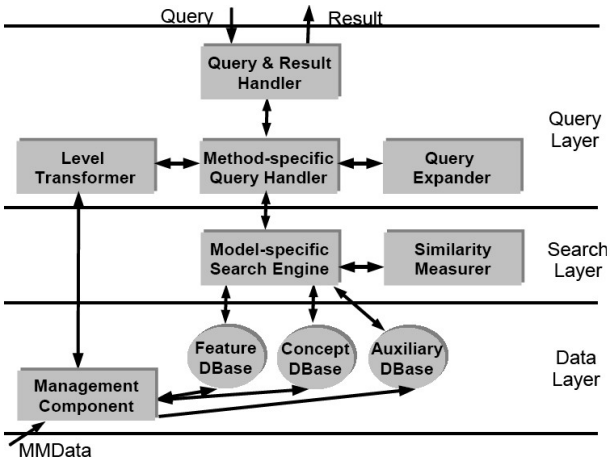


Figure 4.1: Architecture of the Metadata model [3].

4.2 Multimedia Search and Metadata Management

In the paper [3] are distinguished three levels for describing multimedia data.

- The binary level: Raw data without added interpretation (JPEG, MPEG) (raw data module)
- The feature level: Described by a set of features, properties of multimedia data. (color histogram, movement vectors) (metadata module)
- The concept level: Described in terms of semantic concepts (metadata module)

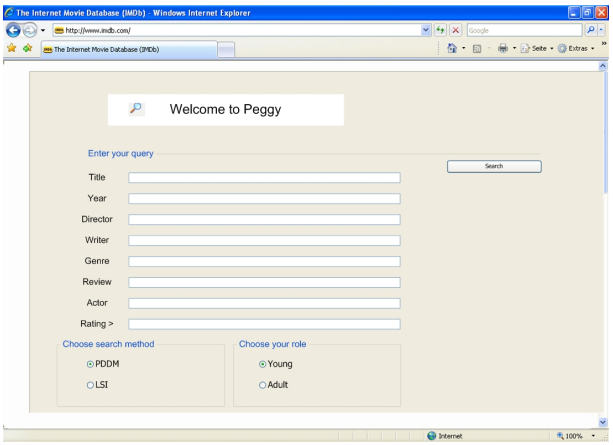
Because the three-level model, there are three kinds of queries. A typical feature query for example is: "Find all images that have a color histogram close to this one", a typical conceptual query is: "Find all multimedia objects containing a car" [3]. The metadata module is the core module in this architecture. In his work they don't want to inspect one solution of query solving or to analyse one specific method. The architecture should be able to support any type of multimedia queries.

A query request has to pass three layers 4.1. In the query layer the query will be breaked down into parts according to the way they should be proceed. Then the query is forwarded to the search engines in the search layer. The search engine is responsible for the actual mapping of query and metadata. Two models are used: The boolean model (classic, based on boolean algebra) and the vector model (assigning non binary weights to index terms). The search engine is directly connected to the data layer that includes the databases. The **feature database** (solves queries at the feature level, contains feature level descriptions of multimedia objects), **the concept database** (contains conceptual descriptions, domain specific semantics, match queries with metadata in the concept space) and the **auxiliary database** (different view of same metadata, rather time consuming, Latent Semantic Indexing method). The search engine forwards its results of the databases again to the

query layer. When all queries are solved, the *Result Handler* take this into account to give the results.

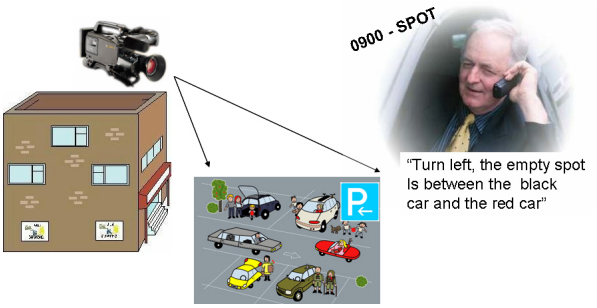
4.3 Case Studies

4.3.1 Peggy



Peggy is a so-called Internet movie search and retrieval service. A user makes a search request about a list of specified parameters in the movie database. As a result he will receive a list of references by complying movies. Movie trailers are stored in the Windows Media Server (Raw data), metadata is saved with the help of the MPEG-7 format. Two query languages are supported as well as two search engines (PDOM [1] and KWEELT [2]).

4.3.2 Spot-a-spot



The spot-a-spot system is a parking guide system. A camera finds free parking lots around a building with the help of colour histograms and identifies the colour of the cars next to the slot. A call of the driver who is searching for the closest parking lot next to the entrance of the building will be responded with the information. This system works with a feature database, a conceptual database and a binary search engine. The result generates a voice message which is then played to the cell phone of the car driver.

4.4 Conclusion

The basic task of this architecture is the ability to provide extendable and scaleable multimedia information management systems. It is easy to modify and adjustable to the personal needs. The examples show the main use of this system. The way how the system fulfills search requests with the help of raw- and metadata databases and how the request is controlled is new compared to other solutions.

5 Conclusion

For the different combination of digital media and their specific requirements and problems no universal architecture can be found. Since for these architectures various approaches exist it is even difficult to give a general overview. We introduced some interesting models of different multimedia architectures that cover a wide area of multimedia applications. SAI is a sound/graphical approach, FRAN is animation oriented and MISE handles Interactive Multimedia retrieval requests. Every architecture has advantages in the specific area where it is used.

One problem occurring for all architectures is the incompatibility if we exchange one media against another one. This problem is caused by the different operations that each media brings along. There is no solution for this problem available and it is questionable if it ever will be.

References

[1] GMD-IPSI, The GMD-IPSI XQL engine Version 1.0.2. http://xml.darmstadt.gmd.de/xql/index.html, September 1999.

[2] A. Sahuguet. Querying XML in the New Millennium. http://db.cis.upenn.edu/Kweelt, September 2000.

[3] Erik Boertjes, Willem Jonker, and Jeroen Wijnands. Multimedia information services enabling: An architectural approach. In *Proceedings of the ACM Multimedia 2001 Workshop on Multimedia Information Retrieval (MIR-01)*, pages 18–23, New York, October 5 2001. ACM Press.

[4] Elaine Chew and Alexandre R. J. Francois. MuSA.RT: music on the spiral array. real-time. In *Proceedings of the 11th ACM International Conference on Multimedia (MM-03)*, pages 448–449, November 4–6 2003.

[5] Conal Elliott and Paul Hudak. Functional reactive animation. In *ICFP 97*, 1997.

[6] Conal Elliott, Greg Schechter, Ricky Yeung, and Salim Abi-Ezzi. TBAG: A high level framework for interactive, animated 3D graphics applications. *Computer Graphics*, 28(Annual Conference Series):421–434, 1994.

[7] A. R.J. François. Software architecture for immersipresence. Technical Report IMSC-03-001, Integrated Media Systems Center, University of Southern California, December 2003.

[8] A. R.J. François. Sai: Architecting distributed asynchronous software systems. Technical Report IMSC-05-003, Integrated Media Systems Center, University of Southern California, September 2005.

[9] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph H. Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Richard B. Kieburtz, Rishiyur S. Nikhil, Will Partain, and John Peterson. Report on the programming language haskell, a non-strict, purely functional language. *SIGPLAN Notices*, 27(5):R1–R164, 1992.

[10] John M. Snyder. Interval analysis for computer graphics. *Computer Graphics*, 26(2):121–130, 1992.

[11] Wikipedia. First-class object — wikipedia, the free encyclopedia, 2006. [Online; accessed 24-January-2007].

[12] Wikipedia. Multimedia — wikipedia, the free encyclopedia, 2007. [Online; accessed 18-January-2007].