SAI: Software Architecture for Immersipresence

1. Research Team

Project Leader: Prof. Alexandre R.J. François, *Computer Science*

Graduate Students: Cheng Zhu

2. Statement of Project Goals

The goal of the SAI (Software Architecture for Immersipresence) project is to provide a universal framework for the distributed implementation of algorithms and their easy integration into complex systems that exhibit desirable software engineering qualities such as efficiency, scalability, extensibility, reusability and interoperability. SAI is supported by MFSM (Modular Flow Scheduling Middleware; mfsm.SourceForge.net), an open source architectural middleware implementing the core SAI architectural primitives.

3. Project Role in Support of IMSC Strategic Plan

SAI is a software architecture model for designing, analyzing and implementing applications performing distributed, asynchronous parallel processing of generic data streams. SAI provides formal, system design-oriented architectural support compatible with agile methodology practices. In particular, SAI's modularity facilitates incremental system design, development, maintenance and evolution. The underlying asynchronous parallel processing model ensures that optimal system throughput and latency are achievable for a given design. The open source MFSM project provides a common code base for independent (but concerted) development of processing modules and their integration into complex applications and systems. Focused experiments in complex system integration support specific research efforts, and may serve as models for larger scale integration efforts.

4. Discussion of Methodology Used

SAI specifies an architectural style [9], whose underlying extensible data model and hybrid (shared memory and message-passing) distributed asynchronous parallel processing model allow natural and efficient manipulation of generic data streams, using existing libraries or native code alike. The modularity of the style facilitates distributed code development, testing, and reuse, as well as fast system design and integration, maintenance and evolution. A graph-based notation for architectural designs allows intuitive system representation at the conceptual and logical levels, while at the same time mapping closely to the physical level.

MFSM (Modular Flow Scheduling Middleware; mfsm.SourceForge.net) is an open source architectural middleware implementing the core elements of the SAI style. MFSM aims at promoting and supporting the design, analysis and implementation of applications in the SAI style. A number of software modules regroup specializations implementing specific algorithms or functionalities. They constitute a constantly growing base of open source, reusable code,

maintained as part of the MFSM project. The project also comprises extensive documentation, including user guide, reference guide and tutorials.

Figure 1 shows the overall system architecture suggested by MFSM. The middleware layer provides an abstraction level between low-level services and applications, in the form of SAI software elements. At the core of this layer is the Flow Scheduling Framework (FSF), an extensible set of foundation classes that implement SAI style elements. The generic extensible data model allows encapsulating existing data formats and standards as well as low-level service protocols and APIs, and making them available in a system where they can interoperate. The hybrid shared memory and message passing parallel processing model supports control, asynchronous concurrent processing and synchronization of data streams. The application layer can hosts a data-stream processing software system, specified and implemented as instances of SAI components and their relationships.

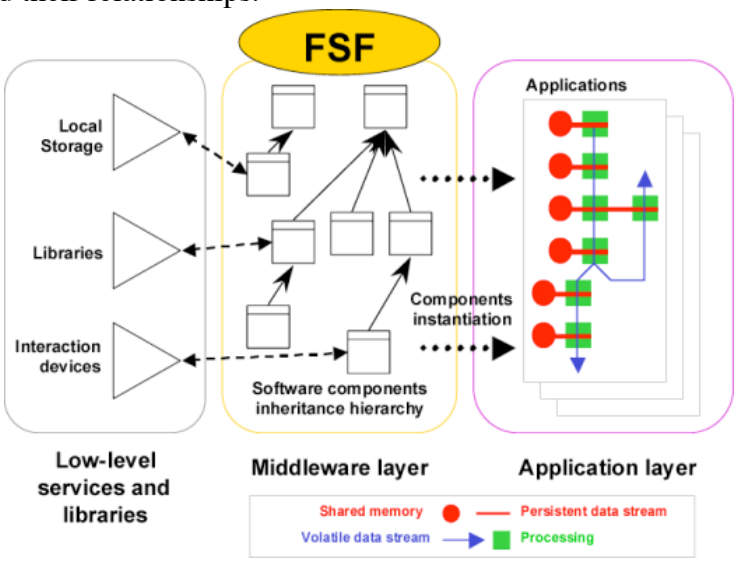

Figure 1 - MFSM system architecture model.

5. Short Description of Achievements in Previous Years

In the starting year (1999-2000), we introduced the layered software architectural framework that would evolve into SAI. We defined the FSF [4], cornerstone of the framework, by specifying data and processing models. We developed a first prototype implementation with some functional modules to demonstrate the first real-time data stream processing system designed according to the model, the "Blue Screen Without a Blue Screen" segmentation demonstration.

In 2000-2001, we introduced a preliminary design for an interactive application development and execution environment. Concurrently, we validated the SAI concepts and refined the supporting implementation by integrating new functionalities in the code base (core library and supporting libraries). We demonstrated improved performance for the segmentation application [5], and demonstrated the "Video Painting" real-time mosaicing system.

In 2001-2002, we launched the MFSM open source project, hosted on SourceForge.net [3]. MFSM is an architectural middleware implementing the defining elements of the SAI framework. The initial releases included the core library implementing FSF, some supporting

modules and libraries, documentation and tutorials. A module implementing a scene graph structure was also developed, and used to demonstrated live video mapped onto an animated 3-D model in an interactive setting [7]. Other projects developed using MFSM included the "Virtual Mirror" [6].

In 2002-2003, we created a first of its kind course dedicated to media systems integration. The high point of the course was the realization of an ambitious class project, whose collaborative distributed development was made possible by the use of SAI and MFSM. We also started a number of focused exploration projects, with the goal of developing support for key features in MFSM. Feedback from teaching and research projects steered evolutions of the MFSM open source implementation and package design (documentation, demonstrations and tutorials), and of the SAI framework itself.

5a. Detail of Accomplishments During the Past Year

The software design and development principles comprising SAI, so far explored experimentally, were embedded into a well-defined and academically accepted systematic framework for the description and study of software architectures [14]. Placed in this context, SAI defines a new component-based architectural style [9]. This formalization work provides the grounds for the use of theoretical and practical tools for designing and evaluating software architectures, predicting and analyzing their performance, and implementing them on specific platforms. Although this line of exploration is only beginning, new such tools will certainly be developed along the way.

The first direct applications of this formalism are in the development of an Integrated Visual Architecture Design and Analysis Environment for SAI. The first prototype developed this year focuses on the visual composition of SAI components into architectures whose validity with respect to SAI rules can be verified and enforced. This environment is a first step towards an Integrated *Perceptual* Architecture Design and Analysis Environment, which will be designed with SAI.

The MFSM open source project has undergone significant growth, with regular updated releases of the base FSF library, completion of reference and user guides, release of several open source modules with documentation and examples, and thematic tutorials.

The availability of a significant amount of quality online material, together with scientific publications in relevant research communities, either in process (e.g. [9.10]) or in the planning (e.g. [8]), contribute to enhancing the visibility of the SAI project not only at IMSC but also in the research community and among current and potential industrial partners.

In order to provide further instruction and support to current users and potential adopters, we have started an IMSC SAI Workshop series (http://iris.usc.edu/~afrancoi/sai/workshop), hosted at USC. The workshop is intended for (1) system designers and developers who are current or potential SAI users, (2) Decision makers who are in need of an adapted software engineering framework, and (3) potential SAI project contributors. These functional categories comprise students and faculty, as well as researchers and developers from academia and industry. The first

workshop took place in early October 2003. Over 20 IMSC researchers attended the two half-day-event, which offered an introduction to SAI spanning theoretical foundations, integrated system design, and code development using the open source middleware MFSM. A second workshop is scheduled for the end of the Spring 2004 semester.

Intrinsic developments in SAI and MFSM are supported and steered by focused integration experiments conducted simultaneously, resulting in a wider range of functionalities in MFSM modules, and better understanding of SAI properties and architectural patterns. These experiments resulted in the development of integrated systems that support specific research efforts, and may serve as models for larger scale integration efforts. One such experiment is the MuSA.RT project (http:// cappriccio.usc.edu/MuSA.RT), which is described in a separate Volume Two report. We highlight here another integration experiment, the IMSC Communicator project, which started last year in the form of a set of studies on communication and audio/video input and output modalities.

The IMSC Communicator (http://iris.usc.edu/~afrancoi/imsccommunicator) is an experimental extensible platform for remote, collaborative data sharing. The goal of this new project is to explore and formalize architectural patterns for true multi-stream systems. At the core of the Communicator is a sequence of cells introducing network communication between two independent subgraphs. This communication pattern is used as a generic platform for designing and testing data transfer modalities, synchronization schemes and corresponding architectural patterns. Figure 3 shows the architecture and some screen shots of a version of the IMSC Communicator combining video communication (image and sound) and a real-time background replacement unit based on the “blue-screen without a blue screen” developed at IMSC. Background replacement was added to the capture side to illustrate how the modularity of the architecture allows to ``plug-and-play'' subgraphs developed independently.

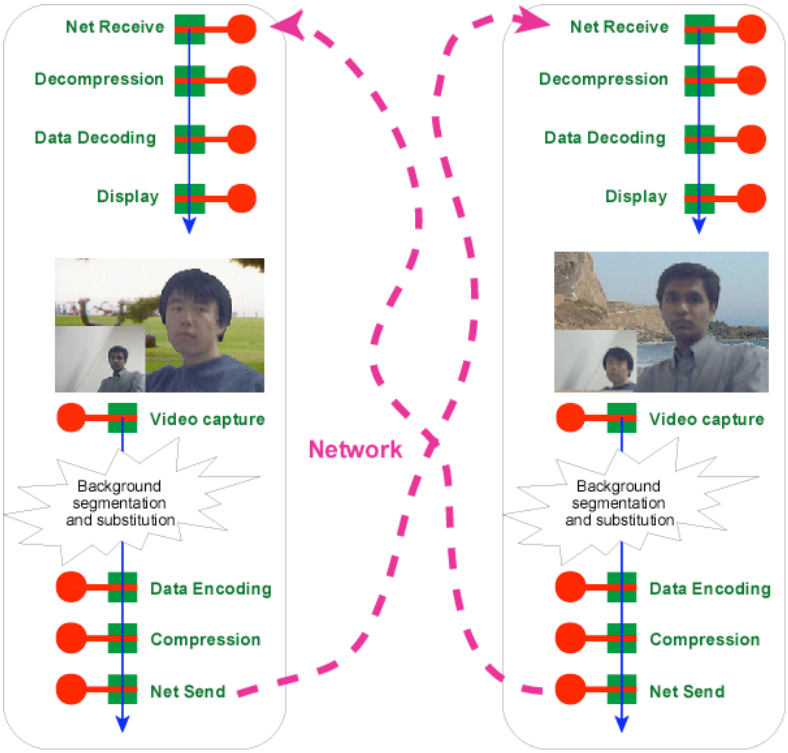

Figure 3 - A version of the IMSC Communicator with support for video data.

6. Other Relevant Work Being Conducted and How this Project is Different

The issue of (software) system integration is addressed only partially, when at all, in individual fields. Software libraries emerge in specific fields as they reach the necessary levels of maturity. Examples include OpenGL, and Microsoft's set of Multimedia libraries DirectX. Standard libraries are extremely useful because they provide a common set of data structures and algorithm implementations that can be re-used across applications. As they are designed with a relatively narrow field of application in mind, they do not address interoperability. In particular, such libraries might be built explicitly or implicitly on architectural models that are incompatible.

Multimedia: The dramatic development of networks and the Internet made ``multimedia'' the field of research dealing essentially with storage, retrieval, transmission and presentation of large amounts of data. The multi- prefix is justified by the handling of data that might not be text (e.g. images and sound), and that is usually in the form of bandwidth-intensive data streams. The goal is to deliver high quality media. Because of the predominance of the communications aspect, system architecture in this context usually refers to the underlying networking topology (e.g. client-server or peer-to-peer). These network-centric architectural concerns however do not address the internal organization of the various components, which is usually where the major cross-disciplinary issues arise. Bandwidth being regarded as the major limiting factor, the only on-line (real-time) data processing performed is related to compression and decompression. Presentation does not involve much processing beyond decompression and display, which justifies the adoption of dataflow-based approaches in DirectShow and all other so called

``media streaming'' libraries and packages, in research and industry alike. Example research projects include MIT's VuSystem [11], the Berkeley Continuous Media Toolkit [13], the Network Integrated Media Middleware [12], and the Distributed Media Journaling (DMJ) project [1]. These efforts all rely on modular dataflow architecture concepts. They are designed primarily for audio and video on-line processing and transmission, with a strong emphasis on capture, transmission and replay aspects. They cannot easily scale up to applications involving immersion, interaction and synthetic content mixing.

Interactive games: Distributed, first person collaborative games have probably more in common with the Immersipresence vision than any other type of multimedia systems. This stems from the fact that games are first and foremost highly interactive applications. Where video-on-demand and other high-fidelity media streaming applications are yet to become business realities, gaming is actually an already profitable, and fast growing industry (US$6.9 billion in US sales in 2002 according to the Entertainment Software Association), subject to market imposed constraints. The most advanced designs (including architectural designs) are proprietary, and constitute valuable intellectual property for game development companies. Game development projects have become multi-year team productions, with multi-million dollar budgets, and industrial-size management problems. More importantly, game engines are by design extremely fine-tuned systems with very specific hardware constraints that do not necessarily scale-up for architectural generalization. Furthermore, time constraints in new developments usually translate into quick fixes rather than re-design of existing software. Various commercial so-called middleware packages are available, that provide support for specific aspects such as physics simulation, artificial intelligence, networking, etc. Their use in particular projects is determined by the amount of work needed to interface them with a given game engine [2.3], which often outweighs the advantages of reuse. This reinforces the view that game engines are not built as modular, extensible platforms, but rather fine-tuned, ad-hoc systems designed to get the most out of a specific target hardware configuration. This approach is clearly not suitable for research projects. On the academic side, game design is only very slowly making its way as a valid engineering research topic, and thus the development and study of generic architectures is not a mainstream effort.

Software architecture: It is interesting to notice that the set of issues raised in the development of modern games is largely disjoint from that of issues raised by the development of multimedia streaming systems. Although core technologies might find applications in both fields, they are two totally distinct efforts at the system architecture level. Yet, both efforts provide (partial) technical solutions, along different paths (higher bandwidth vs. higher interactivity) towards the Holy Grail of Immersipresence, as shown in Figure 4.

Traditional architectural styles [14], including those with higher level of abstraction, have well-defined and focused areas of optimal relevance, so that a given style is usually not suitable for all aspects of a given complex system. Most systems are thus designed in different styles, introducing unmapped areas at the interface between the resulting parts. In order to allow consistent design and modeling of complex systems, a new, unifying hybrid style is needed. SAI was developed for this purpose.

7. Plan for the Next Year

Our plan for next year is to continue our groundbreaking research in media systems integration:

- Formal study of SAI; creation and development of theoretical and practical tools for architectural design and analysis of complex systems; development of an Integrated Visual Architecture Design and Analysis Environment.
- Continuation of current experiment and demonstration projects, creation of new ones as necessary to support exploration of architectural patterns for Immersipresence and their properties.

8. Expected Milestones and Deliverables

- First version of the Integrated Visual Architecture Design and Analysis Environment.
- MFSM updates, new modules and tutorials.
- IMSC SAI Workshops.

9. Member Company Benefits

SAI provides a uniform framework for the design of integrated media systems inside IMSC or in cooperation with industrial partners. The SAI style allows fast prototyping for proof-of-concept demonstrations, and may prove to be a valuable framework to complement modern software engineering practices. SAI is well suited for distributed development of functional modules, and their seamless integration into complex systems. The modularity of the design allows gradual development, facilitating continuing validation and naturally supporting regular delivery of incremental system prototypes. By enabling the use of software engineering metrics for project management (e.g. planning and quality control), using the SAI style in research projects may facilitate technology transfer to industry. In industrial R&D environments, the SAI style allows fast prototyping for proof-of-concept demonstrations.

10. References

[1] Eide V.S.W., Eliassen F., Granmo O.-C. and Lysne O., "Scalable Independent Multi-level Distribution in Multimedia Content Analysis," Proceedings of the Joint International Workshop on Interactive Distributed Multimedia Systems / Protocols for Multimedia Systems (IDMS/PROMS 2002), Coimbra, Portugal, 2002.

[2] Dybsand E., "AI middleware: Getting into character" (parts 1-5). www.gamasutra.com, July 2003.

[3] Ferguson M. and Ballbach M., "Product review: Massively multiplayer online game middleware." www.gamasutra.com, January 2003.

[4] François A. and Medioni G., "A Modular Middleware Flow Scheduling Framework." Proc. ACM Multimedia 2000, pp. 371-374, Los Angeles, CA, November 2000.

[5] François A. and Medioni G., "A Modular Software Architecture for Real-Time Video Processing." Proc. International Workshop on Computer Vision Systems, Vancouver, Canada, July 2001.

[6] François A., Kang E. and Malesci U. "A Handheld Virtual Mirror," SIGGRAPH Conference Abstracts and Applications proceedings, p. 140, San Antonio, TX, July 2002.

[7] François A., “Components for Immersion,” Proceedings of the IEEE International Conference on Multimedia and Expo, Lausanne, Switzerland, August 2002.
[8] François A. Software Architecture for Immersipresence. IMSC Technical Report IMSC-03-001, December 2003.
[9] François A., “A Hybrid Architectural Style for Distributed Parallel Processing of Generic Data Streams,” Proceedings of the International Conference on Software Engineering, Edinburgh, Scotland, UK, May 2004.
[10] François A., “Software Architecture for Computer Vision,” Emerging Topics in Computer Vision, G. Medioni and S.B. Kang Eds., Prentice Hall, 2004.
[11] Lindblad C.J. and Tennenhouse D.L., “The VuSystem: A programming system for compute-intensive multimedia,” IEEE Jour. Selected Areas in Communications, 14(7):1298-1313, September 1996.
[12] Lohse M., Repplinger M. and Slusallek P., “An Open Middleware Architecture for Network-Integrated Multimedia,” Proceedings of the Joint International Workshop on Interactive Distributed Multimedia Systems / Protocols for Multimedia Systems (IDMS/PROMS 2002), Coimbra, Portugal, 2002.
[13] Mayer-Patel K. and Rowe L.A., “Design and performance of the Berkeley Continuous Media Toolkit,” Multimedia Computing and Networking, M. Freeman, P. Jardetzky and H.M. Vin Eds., pages 194-206. 1997.
[14] Shaw M. and Garlan D. Software Architecture - Perspectives on an Emerging Discipline. Prentice Hall, Upper Saddle River, NJ, 1996.