Interactive Multi-User Multimedia Environments on the Internet: An Overview of DAMSEL and Its Implementation

Paul Pazandak, Jaideep Srivastava
Distributed Multimedia Center
Department of Computer Science, University Of Minnesota
pazandak|srivasta@cs.umn.edu
http://www.cs.umn.edu/~pazandak/damsel.html

Abstract

This paper presents an overview of DAMSEL and it's implementation. DAMSEL is comprised of an embedable dynamic multimedia specification language, and a software architecture suitable for multi-user interactive multimedia environments. The goal of DAMSEL is to explore language constructs and execution environments for next-generation interactive multimedia applications. The constructs of DAMSEL include primitives for event-driven temporal specification – supporting causation and inhibition. Specifications allow behavioral parameters to be chosen, enabling very powerful temporal relations to be defined. Other constructs support the modification and analysis of multimedia data, and high-level abstractions for connections to user interfaces. Further, DAMSEL constructs support conditional and constraint logics, enabling more complex specifications than currently possible. DAMSEL is being implemented using a java implementation of the CORBA standard.

1. INTRODUCTION

To date, several research projects (such as [1]) and commercial products have focused on providing languages and applications to develop multimedia presentations. These solutions enable a user to produce non-interactive and sometimes hypermedia-based or interactive presentations. Most of these provide a graphical development environment, or a scripting language to define the presentations. However, for an application developer integration into other application environments which are developed using C, C++, and now Java, for example, can be difficult. In addition, most (if not all) of these solutions produce only single-user presentations that simply allow the viewing of multimedia data.

DAMSEL, a **D**yn**A**mic **M**ultimedia **S**p**E**cification **L**anguage, enables developers to more easily write multi-user interactive and collaborative applications that can retrieve, view, modify, analyze, and store multimedia data. DAMSEL provides these features using three language components and an underlying software architecture: the dynamic event-driven temporal component, providing interaction-driven media and event orchestration mechanisms; the dataflow component, handling retrieval, modification, analysis and storage; and, the presentation component, handling the presentation/viewing of multimedia. DAMSEL is composed of just a few predicates so that it can be easily embedded within another language, eliminating the need to learn and use an entirely new language, compiler, and programming environment. The software architecture supports the execution of DAMSEL statements using the run-time event manager, dataflow manager and presentation server. A portion of this architecture is currently being implemented using a java-based CORBA product, Post Modern's BlackWidow. This software will allow us to execute DAMSEL across the internet on all platforms that support java.

The goal of DAMSEL is to explore multimedia language constructs and supporting execution environments for next-generation interactive multimedia application development. The basic features of DAMSEL include declarative specifications (strictly non-procedural), expressiveness, simplicity, conditional and constraint logics, extensible behavioral specification parameters, programming language-embedable, and an open systems approach enabling current software to be integrated. In addition, we are investigating a subclass of complex sequence-based event detection (time-sequence detection) and the creation of a time-

0-8186-7436-9/96 $5.00 © 1996 IEEE
Proceedings of MULTIMEDIA '96

sequence event definition language to detect motion-based user interaction.

In the following three sections we present an overview of each language component; this is followed by a discussion of the current architecture.

2. Temporal Component

As described above, DAMSEL has three language components that provide high-level constructs for creating interactive applications capable of retrieving, modifying, analyzing, viewing and storing multimedia data.

The foundation of these is the event-driven temporal language component. Specifications can be defined by users and application developers to represent the behavior of the system. Each statement within a specification defines either an excitatory or inhibitory relationship between some set of events. Simply, when one event occurs, it excites (or causes) another event to occur (or inhibits another event from occurring). Since system, application and user interactions are interpreted as events, the actual behavior of the system is determined only at run-time as occurring events cause other events to be generated, based upon the specifications (which can be submitted at run-time). In addition, the behavior of each statement within a specification is dictated by an extensible set of behavioral parameters. It is this, in part, that enables DAMSEL to express all of the temporal relationships expressible in any of the sixteen other models we compared [3].

To orchestrate the delivery of media objects, a *temporal specification* is defined; its structure is regulated by a temporal specification model. There is a wide range of proposed (and implemented) temporal specification models to date [5] – timeline, hierarchic, synchronization point, and more recently, event-driven. At one end of the spectrum, a temporal specification model may be static, only allowing the time of delivery to be tied to a clock. At the opposite end, a model such as DAMSEL has, will support dynamic interaction-driven specifications by enabling a *behavior* of the system to be specified, while the actual execution (different for each user) will depend upon system, application, user-media, and user-user interactions (as well as user profiles, user performance, resource availability and data loss, for example).

In addition, a finer level of synchronization is required to coordinate the presentations of media objects that have a high degree of *temporal interdependency* to support lip-synchronization, for example.

The temporal component of DAMSEL is based upon an event-driven approach and includes two simple, yet powerful relations for expressing *activation*, *inhibition* and *fine-grain synchrony*. The rest of this section provides a brief overview, while a more extensive description can be found in [3].

We implement activation and fine-grain synchronization using the predicate "causes", and inhibition using "defers". DAMSEL also addresses static and dynamic specification conflict resolution [4].

Within DAMSEL, we define causality (activation) using two events, x and y, such that "the occurrence of event x causes the occurrence of event y," where event x is the *trigger* and event y is the *bullet* associated with some action that will be invoked. Causal relations are defined using the predicate *causes*. It takes two events, EVTx and EVTy; an optional range delay interval, d (d_i,d_j); an optional set of system-defined extensions (behaviors), and a statement name, s_1:

s_1::causes (EVTx, EVTy, d, {set-of behaviors})

The basic specification should be interpreted as: "The occurrence of EVTx will cause the occurrence of EVTy." EVTy will occur at [the occurrence time of EVTx + a valid value within the range d], executed using the specified set of behaviors. Note that EVTx can be defined as *any* event or condition composed of DAMSEL's conditional logic and standard logical and relational expressions using global variables. In addition, EVTy can be a *set* of events.

While activation brings about the occurrence of an event, in DAMSEL we define a means to inhibit (or defer) an event from occurring. Deferment could be thought of as an inhibitory synapse which is applied to a neuron (event) to inhibit it from firing, while causation is similar to an excitatory synapse which causes a neuron (event) to fire. In DAMSEL, deferment is specified using the *defers* predicate. It takes one event, EVTy, an interval event INTa, an optional delay value, D, (default = 0), an optional set of system-defined extensions (as described earlier), and a statement name. It has the following form:

s_2::defers (INTa, EVTy, D, {set-of behaviors})

This specification should be interpreted as: "INTa defers EVTy"; or, not quite so terse as: "if event y would occur during interval INTa, the occurrence of event EVTy will be deferred *at least* until after the occurrence time of interval INTa's end event (+ delay value D)." As above, EVTy can be a *set* of events to be deferred. In addition, INTa can also be specified as any two bounding events, EVTa and EVTb, which describe some interval.

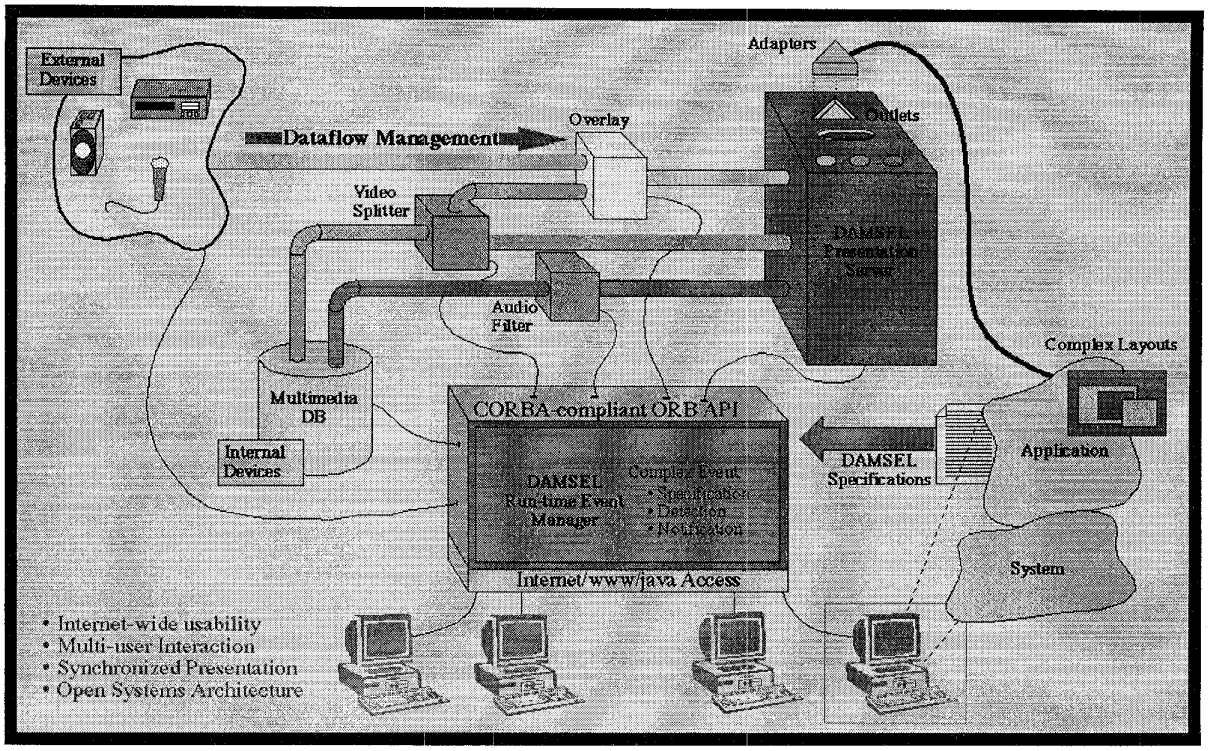

Figure 1. An Overview of the DAMSEL Project

3. Dataflow Component

The dataflow language component [4] is based upon a stream model, whereby the media objects are modeled as continuous streams, flowing from sources to sinks. The sources can be storage devices, or live device sources such as microphones and video cameras; while examples of sinks may include graphical layouts, external processes, and storage devices. The streams can be modified, and analyzed en route to the sinks by defining paths that include such operations which the streams must pass through (see Figure 1). Other work such as [2] have also used this basic model augmented with graphical interfaces. The language of this component would also be suitable for such an interface.

A specification defines the order in which these operations should be connected together to create the path which the stream will follow. In DAMSEL, sources have outlets, sinks have inlets, and operations (or source-sinks) have inlets and outlets – these are all classified as stream objects. Each stream object may have one or more inlets and/or outlets, which is specified as part of an object's definition. Therefore, a specification must indicate which outlets and inlets are connected together. If the objects being connected together each only have one inlet and outlet respectively, then they do not need to be indicated. The basic syntax describes one connection (using minimal canonical form):

$$s_i :: streamObj_i.name(outlet_m) \rightarrow streamObj_j.name(inlet_n)$$

where each $streamObj_i.name$ is a specific instantiation of a stream object of type $streamObj_i$.

4. Presentation Component

The presentation language component supports specifications which control the connection and delivery of streams to consuming processes, windows, and devices via *adapters*. These may include display device cards and complex layouts defined by graphical toolkits. The presentation component includes a server (acting as a dataflow sink) to which streams are connected in the dataflow component – any number of streams can be connected to a presentation server (see Figure 1).

Within the presentation server, "complex outlets" can be defined using any combination of streams that are attached to the server. Complex layouts can be thought of as *views* that are used within database ap-

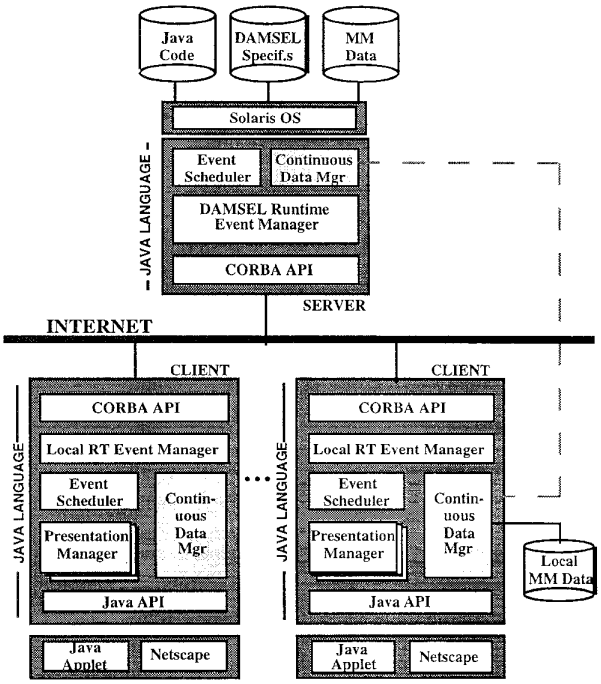

Figure 2. DAMSEL's Execution Environment

plications – allowing the user to select a view appropriate for the context in which the data is being presented. Layouts and devices connecting to the server then use compatible adapters to make connections with server outlets. Adapters can be mated with any outlet whose composition matches the adapter's composition. Since specifications from the dataflow and presentation components can be used within specifications from the temporal component, DAMSEL also supports dynamic dataflow and presentation specifications

5. Implementation

Currently, we are implementing the temporal component and the underlying software architecture using a java-based CORBA environment. This architecture is suitable for multiple collaborating users (see Figure 2), while a single client could be used for standalone applications. All client-server and client-client communications are handled by the CORBA services. For master-slave applications, the server run-time event manager controls the overall execution through statements (and additional server code if desired). Clients send important events to the server. A client's local run-time event manager handles localized interaction with the application, and client-client interaction. Client-directed statements execute locally and do not need to communicate with the server. Each run-time event manager manages it's set of statements, and any new statements submitted to it. Simple or complex events and conditionals associated with a trigger are detected using an optimized object-based implementation of the GridMatch algorithm [6] (defined for AI production systems) which allows sharing of subcondition evaluation across multiple statements. When a statement's trigger becomes true, the bullet is fired and the associated events are sent to the event scheduler for execution (using the set of specified behaviors).

The presentation manager controls the delivery of multimedia to the application or screen. The continuous data manager is not currently being implemented due to throughput constraints of the internet. Instead, continuous media can be downloaded to the client ahead of time for presentations using video. Finally, the client application can be written in java to define the user interface and additional application code. This architecture would be suitable, for example, for internet-based multi-user assisted learning environments with self-exploration, and small group interaction and collaboration. A more detailed description of our implementation will be available at our website, and in an upcoming paper.

References

[1] Drapeau, G.D. and H. Greenfield. "MAEstro - A Distributed Multimedia Authoring Environment," in *USENIX*. 1991. Nashville, TN.

[2] Gibbs, S. "Application Construction and Component Design in an Object-oriented Multimedia Framework," in *Network and Operating Systems Support for Digital Audio and Video. Third Int'l Workshop Proceedings*. 1992. Germany.

[3] Pazandak, P., J. Srivastava and J. Carlis, "The Temporal Component of DAMSEL," *Second Workshop on Protocols for Multimedia Systems (PROMS '95)*, Salzburg, Austria, 1995.

[4] Pazandak, P., J. Srivastava, "The Language Components of DAMSEL: An Embedable Event-driven Declarative Multimedia Specification Language" *SPIE: Electronic Imaging International Conference, Photonics East '95*, Pittsburg, PA., 1995.

[5] Steinmetz, R., "Synchronization Properties in Multimedia Systems," *IEEE Journal on Selected Areas in Communications*, 1990. 8(3): p. 401-412.

[6] Tan, J.S.E., M. Maheshwari, and J. Srivastava, "GridMatch: A Basis for Integrating Production Systems with Relational Databases," *IEEE Conference on Tools with AI*, 1991.