# An Active Events Model for Systems Monitoring

**Philip N. Gross, Suhit Gupta, Gail E. Kaiser,**
**Gaurav S. Kc, Janak J. Parekh**
{png3, suhit, kaiser, gskc, janak}@cs.columbia.edu

Programming Systems Lab
Department of Computer Science
Columbia University
500 W 120th St.
450 Computer Science Building
New York, NY 10027
(212) 939-7100
*http://www.psl.cs.columbia.edu*

# Abstract

We present an interaction model enabling data-source *probes* and action-based *gauges* to communicate using an intelligent event model known as *ActEvents*. ActEvents build on conventional event concepts by associating structural and semantic information with raw data, thereby allowing recipients to be able to dynamically understand the content of new kinds of events. Two submodels of ActEvents are proposed: *SmartEvents*, which are XML-structured events containing references to their syntactic and semantic models, and *Gaugents*, which are heavier but more flexible intelligent mobile software agents.

This model is presented in light of DARPA's DASADA program, where ActEvents are used in a larger-scale subsystem, called KX, which supports continual validation of distributed, component-based systems. ActEvents are emitted by probes in this architecture, and propagated to gauges, where "measurements" of the raw data associated with probes are made, thereby continually determining updated target-system properties. ActEvents are also proposed as solutions for a number of other applications, including a distributed collaborative virtual environment (CVE) known as CHIME.

# Introduction and Motivation

DARPA's DASADA program has focused on standards for distributed systems to ease assembly and maintenance of systems that are composed of components "from anywhere" (e.g., COTS, GOTS, open source, etc.). This program has focused on four areas: *architecture description languages* to describe the composed system, *probes* to gather information about the current system configuration and state, *gauges* to interpret this information, and *adaptation engines* that can reconfigure the system as necessary.

This paper focuses on the interaction between probes and gauges, and proposes a standard for data interchange between them. The control interfaces for both probes and gauges have been developed extensively, and standards have been proposed by others. However, the format and transmission mechanism for data collected from probes is underdeveloped. We examine the problem and suggest possible models and architectures, along with a description of our implementation and experience using it.

## *Probes, Gauges and Events*

A *probe* is defined as "an individual sensor attached, either statically or dynamically, to a running program" [1]. Probes emit events that describe some aspect of a program's execution, either at a specific point in time or over some duration. Probes usually:

- are integrated into or wrapped onto the application itself;
- communicate with the application via an API; or
- look at indirect measures such as operating system or network resource usage.

The proposed control interface for probes consists of the following methods: Deploy, Install, Activate (and their inverses), Query-Sensed and Generate-Sensed to enumerate the events that a probe can send, and the Sensed method to publish an event. The newer Focus interface allows additional probes to be activated for detailed examination of a problem. The DASADA standard assumes that probe data will be emitted in the form of Siena events.

For the purposes of this paper, we define an *event* as "a collection of data produced by a system component, and of interest to zero or more other system components." Note that this description makes no assertions about formatting, routing, or transport.

The University of Colorado at Boulder's Siena event system [2] enables Internet-scale content-based event delivery. Siena models events as an unordered, flat collection of attribute-value pairs.

*Gauges* [3] are defined as "software entities that gather, aggregate compute, analyze, disseminate and/or visualize measurement information about software systems." Gauges support a simple configuration interface. The proposed gauge standard includes the concept of a "Gauge Reporting Bus," which is specifically for communicating gauge reports to consumers (who might, e.g., authorize repairs). Consumers supply callbacks to the reporting bus, which are called when an event of interest occurs.

## *Probe-Gauge Interaction*

Probes use system-specific techniques to extract data from the target system. Gauges use the Gauge Reporting Bus interface to report to higher-level components. While the respective APIs for Probes and Gauges are clearly specified, there is no proposed standard for formatting probe data and sending it to the appropriate gauges. Since one cannot assume that probes and gauges will be located on the same machine, some form of networked interprocess communication (IPC) is necessary. Since the machines may be of heterogeneous type, the format for probe data should be as portable as possible.

While we do not address the issue in this paper, we also note that the standard interface for controlling probes, although presumably intended to be an event-based interface, is in fact specified as an RPC-style function API.

### *The Problem*

There are three aspects of the probe-gauge relationship that make the problem of connection difficult: the dynamic nature of individual probes, the dynamic topology of the various components, and the heterogeneous nature of the systems involved.

Individual probes may be frequently added and removed from the system. Probes may be heterogeneously sourced, with possibly different semantics for similar-looking data; simply labeling the type of data elements within the event, as in traditional attribute/value pairs, is insufficient. Instead, the semantic information required for proper interpretation of the probe data must be associated with the event.

Probes and gauges will be activated and deactivated, and may migrate from machine to machine. Some of these components (especially probes) may be running on constrained devices, and requiring every component to maintain a complete network topology is not feasible. Further, since the main tasks for most probes are straightforward, requiring all of them to add the data and logic necessary to manage bidirectional RPC with gauges in a changing environment would increase their complexity considerably. Detailed knowledge of event routing and dispatch should ideally be removed from most probes and gauges. While more advanced systems such as CORBA can help with component discovery, probes will typically have many consumers for a single event, which is not handled efficiently under the CORBA model nor under analogous RPC extensions.

The systems involved may be completely heterogeneous with different byte-ordering, operating systems, architectures, etc. Message formatting should be completely architecture independent, and leverage industry standards to the degree possible.

### ***Previous Work***

In the commercial world, a number of event-based systems have been developed for monitoring network and application status, e.g. TIBCO's TIB/HAWK [4] and Tivoli Distributed Monitoring [5]. These are usually oriented towards predefined SNMP events and infrastructure management.

There have been a number of proposed generic event standards from operating system, language, and middleware developers, such as Sun's ToolTalk [6], the OMG's CORBA Event Service [7], and Sun's Java Messaging Service [8]. The first two mechanisms have fairly limited specifications. JMS is an extensive specification, but custom event semantics are only directly supported through messages composed of serialized Java objects.

The most elaborate work is the Rapide/CEP system [9] which provides a full architecture for complex distributed event processing. Rapide/CEP provides for event pattern recognition based on temporal and causal relationships among events. Our work has emphasized maximal runtime reconfigurability and interoperability, at the expense of the large feature set provided by Rapide/CEP.

## ActEvents Model Overview

For these reasons, we propose that probe information should be encapsulated under an "Active Events" model. This model associates semantic information with the event data. This approach has some resemblances to the work on Active Networks in the networking community, in which data packets have additional information or code. We hope to leverage some of that experience while avoiding the mistakes. In particular, the experience of the networking community has shown that models requiring all mobile data items to be intelligent executable code results in unacceptable performance.

Adding some intelligence to events involves balancing two competing constraints. For maximum flexibility and intelligence, one wants lots of extra data (e.g. network topology maps) and sophisticated processing (e.g. execution of mobile code at every routing node). However, many probes will be producing large amounts of very simple data. In that case, the overhead associated with large events and extensive processing is unacceptable.

We propose two separate submodels that together can support most application needs: a lightweight submodel for frequent, similar events and a more sophisticated submodel for more significant events. Both models solve the various problems outlined in the previous section, as described below.

We call the lightweight model *SmartEvents*. These are XML-structured events that contain references to their syntactic and semantic models. The bulk of the sophistication in this model is in a separate parsing engine, leaving the events themselves simple and lightweight.

We call the sophisticated model *Gaugents*. These are intelligent mobile software agents; able to transport themselves around the network and execute code at each location they visit.

The first problem involves the dynamic nature of individual probes and the difficulty in identifying the correct semantic model for interpretation. SmartEvents are interpreted by a special parsing engine that maintains a mapping of grammar to semantics. Gaugents carry the code necessary for correct interpretation.

The second problem is the dynamic topology of the components. SmartEvents uses the Siena system to transport events where they need to go, pushing the problem to the distributed middleware component. Gaugents can dynamically determine their own routing as they move from node to node.

The third problem is the heterogeneous nature of the involved systems. SmartEvents are structured as XML and thus completely architecture independent. Gaugents use system-independent mobile code and require a "receptacle socket" to be present at each participating node for Gaugents transportation and execution.

## *SmartEvent Model*

For frequent simple events, the most efficient technique is to have events contain a reference to their syntactic and semantic model. For these types of events, the event consists of a tagged document where the tags are used not just to indicate the structure of the text but the semantic model under which they should be interpreted.

The *parsing and lookup engine* identifies the "semantic subcomponents" of the event and processes them appropriately. Unfamiliar subcomponents are handled intelligently by sending requests to the data repository.

The *data repository* maps unknown tags to syntactic and semantic information, and delivers this information back to the parsing engine. This component functions as a sort of primitive ontology server.

Once semantic subcomponents have been identified, the lookup engine applies appropriate processing to the events to make them as digestible as possible for later system components. Processing may include augmenting, deleting or rewriting parts of the subcomponent, as well as filtering it entirely.

## *Gaugents Model*

Gaugents may be constructed or parameterized on the fly by a human or a program, then transmitted from host to host using a dynamically determined routing pattern reactive to the latest host's circumstances and surroundings as well as past and planned trajectories. No workflow is required for Gaugents-based ActEvents.

Gaugents can travel between distributed components using their own transportation mechanism. Target system components are equipped with receptacle sockets to enable them to properly receive and execute incoming Gaugents, as well as send them off along their routes. For added dynamism, two important concepts are implemented: execution scheduling of Gaugents at receptacle sockets and route re-configuration.

Execution of Gaugents at receptacle sockets can be specified and controlled precisely; entry and exit conditions can be defined, satisfaction of which is required for Gaugents execution to commence, halt, etc. Other factors like the number of execution iterations can also be preset.

Gaugents' routes are reconfigurable – capable of being modified at any intermediate step. This is useful particularly because it is difficult for the creator of the Gaugent to predict all target components interested in its existence. A solution is to have these target components express their interest indirectly through those components that the Gaugent are likely to visit. Then, the Gaugent can be rerouted to travel to these other components, either through unicast or multicast channels. There are two ways to achieve this: clone the Gaugent by the receptacle socket and retransmit it to the newer target components, or modify the Gaugent's trajectory so as to cover these newer components. The overall effect is advancement towards a pub/sub model for Gaugents.

## Realization of SmartEvents: FleXML

Events are structured as XML messages. We feel that Siena's flat, unordered collection of attribute-value pairs is inadequate for describing common probe results. For example, many probe tools instrument method calls. The data for such an event needs to list the class, object, method, return value, return type, and a variable-length list of parameter type-value pairs. Attempting to interpret such data from an unordered set of attribute-value pairs is needlessly complicated.

Typical SmartEvents will be composed of at least two subcomponents: an outer "envelope" of metadata and a "payload" of specific probe results.

The Siena Internet-scale event network handles event routing purely based on event content, freeing both probes and gauges from the need to manage network topology. Issues in translating between XML-structured SmartEvents and flat Siena events are discussed in the implementation section below.

To support legacy probes, the *Event Packager* component can construct SmartEvents from primitive probe events using custom plug-ins. Additionally, events are copied to persistent stores for later data mining.

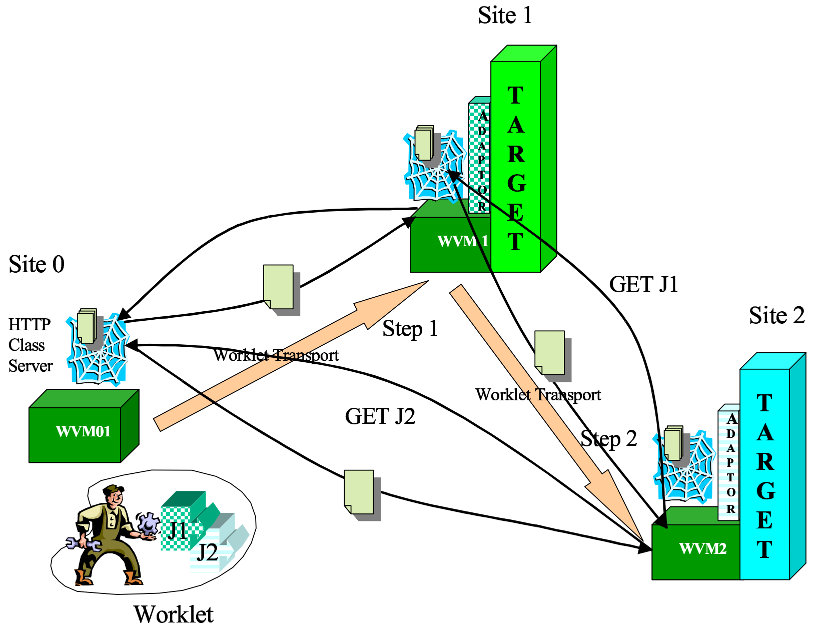
The *Metaparser* does a high-level examination of the event to determine the appropriate subcomponents. Each subcomponent is independently validated, with grammar looked up from the Oracle if necessary. If valid, the appropriate TagProcessor (possibly also retrieved from the Oracle) is applied to the SmartEvent and the result is published.

The *Oracle* maintains mappings of grammar elements to particular syntactic/semantic pairs (e.g., XML schema and tag processor control files). When the Metaparser encounters an unfamiliar tag, it sends the XPath to the Oracle. The Oracle matches the XPath and returns the appropriate schema and control files.

The *tag processor* performs domain-specific processing on the message. This may involve adding, deleting, or rewriting portions of the event, or filtering it entirely. The goal is to present homogenous, simplified events for later processing stages.

### *Realization of the Gaugents Model: Worklets*

Worklets can be defined as self-contained mobile software agents that are deployed on a programmable route of distributed components of a target system, with the purpose of dynamic reconfiguration. The fact that Worklets are self-contained is limited only to the computation that it is performing at the hop in its route; however, in general, the context in which the Worklet executes is actually defined by the local adapter. What this means is that the local adapter provides the missing links required for the sub-general Worklet to execute in the context of the local target system component. The Worklet is therefore contextualized by the target system component it is dealing with.



The *Worklet Virtual Machine* (WVM) is an execution environment for incoming Worklets. It also provides the transportation mechanism that enables the Worklets to travel between successive hops. A host-specific Worklet adaptor must be constructed for each anticipated host system or component, and is attached to that host. In this example, the source of the Worklet is Site 0 with later hops being Site 1 and Site 2. The Worklet travels to Site 1, where the corresponding Worklet junction is scheduled for execution by the local WVM. After execution, the Worklet uses the local WVM to propagate to the following hop in its route.

A separate computation, or *Worklet junction*, is defined for each step along the Worklet's predetermined route. However, this route can be dynamic since it is modifiable on the fly by other Worklets or target systems en route. For non-trivial cases, a *Worklet Jacket* for the current junction determines the customizable scheduling for the Worklet such as pre- and post-execution conditions. The Jacket can also be engineered to let the Worklet continue on its route as soon as it deposits the Worklet Junction instead of waiting for it to complete execution. As mentioned above, the trajectory of the Worklet can be modified dynamically at intermediate WVMs. The Worklet Jacket regulates the extent to which this information, along with other data in the Worklet, is accessed/modified from outside the Worklet.

# SmartEvent Implementation

## *Event structure*

Because particular probes will presumably be updated more frequently than basic metadata, our implementation uses a technology we developed called “FleXML” to allow the description of events through several composed schema fragments. The schemas for Envelope and Payload can thus be managed independently.

The standard metadata envelope for a SmartEvent contains information that will be of interest for all probe events:

- A locally unique identifier;
- The IP address and port of the generating probe;
- A timestamp.

The AIDE system [10] generates information about method calls. Its payload includes:

- Object;
- Class;
- Method;
- A list of type-value pairs for the parameters.

## Metaparser

One of our reasons for choosing an XML-based structure for our messages was the rich set of standard tools available for working with XML. We wanted our parser to leverage existing work as much as possible. However, the APIs of existing parsers are inadequate to handle the style of processing (i.e., use of multiple schemas for a single message) required for the Metaparser.

The fundamental problem of working with composed schemas was solved by constructing an elaborate front end for the processor, which could send different portions of a message to different XML processor instances. To increase performance, we also modified the XML processor to allow caching of parsed schemas.

The Metaparser has a three-layer system for parsing incoming messages according to composed schemas. When a new message arrives, a separate parsing thread begins to examine it. If a tag matches a known semantic subcomponent, a validator is started to handle that subcomponent and pointed at the correct schema. Each validator, in turn, is a wrapper around an Apache project Xerces XML parser/validator. Validators allow particular information to be selectively passed through to underlying parsers. The top-level parsing thread informs the validators which parts of the message are relevant to their schema. Validators pass appropriate data through to their parsers that do the actual XML schema validation.

Some additional complications arise. For instance, the Xerces parser expects complete messages for validation, not fragments. Therefore, extra information is added to the data stream so that the fragments appear to be complete messages to the validating parsers.

Since there will be many messages, most of which will be based on only a small number of schemas, efficient caching of schema information is key to performance. Unfortunately, the current Xerces implementation is rather inefficient, requiring the schema to be read and parsed for each message. We modified the Xerces API to make parsed schemas a first-class data object, similar to the corresponding Oracle implementation. Schemas can now be parsed once, and then repeatedly applied to messages.

## Oracle

The Oracle component uses an SQL database to map XPaths and tags to associated files. It supports both XPaths anchored at the root and “free-floating” context-independent tags. The Oracle waits for request events to arrive, and attempts to match the unknown XPath or tag. If a match is found, a success event is published, and a Worklet is dispatched containing the associated files. Otherwise, a failure event is published.

To load the database, the Oracle provides a graphical interface. The user specifies an XML schema file. The Oracle then parses this file, identifying unique tags. These are then presented as the first column of a table. In the other columns, the user can specify the files that should be sent if this tag is queried.

### *Tag processor*

The TagProcessor applies domain-specific processing to the message. The primary mechanism for doing so is an engine that is controlled by two XML-formatted files.

The first file is a standard XSLT template file that allows arbitrary transformation of the message. Based on analysis purely local to this message, tags can be added, removed, or rewritten.

The second file allows some context-based adjustments to the message. The Metaparser has the option of passing an "environment" symbol table to the TagProcessor. The message can be conditionally modified based on values in the table, and new or modified values can be written to it. This allows one to e.g. maintain a count of a certain type of event.

### *Siena*

The Siena system has a number of features that make it well-suited to the needs of the SmartEvent system. Most importantly, it has a scalable implementation of content-based addressing. This is tremendously valuable for the SmartEvent model, as otherwise gauges and probes would be orders of magnitude more complex. Additionally it has a simple, well-documented interface. The system is also lightweight enough to make integration of wireless handheld devices feasible.

There are a number of problems as well, however. Siena does one-time best-effort delivery. There are situations where store-and-forward at an intermediate node might make the system more robust, as well as enabling intermittently-connected devices to participate.

More significantly, we had to find a system for mapping our XML-formatted messages to Siena attribute-value pairs. We considered flattening the XML, with each attribute consisting of the full XPath to the associated tag. We rejected this as unwieldy.

The currently implemented solution is to put the entire XML message into a single value, and "promote" some of the envelope metadata to the attribute-value level for content-based routing purposes.

## Gaugent/Worklet Implementation

Worklets can implement the local aspect of decentralized workflow [11]; however, we only discuss Worklets with respect to ActEvents here. Worklets provide computational intelligence to ActEvents. In the SmartEvent version of ActEvents, the pub/sub transportation mechanism is suited for general, frequent event notifications. However, there are other cases where bi-directional streaming of *data-only* notification events is not effective, e.g., for those relatively infrequent circumstances where processes need to run at the recipients of the events.

Consider a scenario where the KX system is monitoring a mission critical system, e.g., a manufacturing control system. If one of the probes detects a critical condition, it could send out "executable" SmartEvent-based ActEvents to notify KX of the problem, and KX would eventually send a fix into the target system.

A better solution is to have the probe issue a Worklet-based ActEvent that can directly carry out remedial tasks immediately without waiting for KX. This allows emergency reactions to execute immediately without waiting for the higher-latency KX to identify the problem, determine a response, and dispatch a Worklet.

This can be interpreted as being complementary to standard RPC-style communication, with all callbacks being handled not by the sender of the ActEvent, but by its mobile proxy (the ActEvent itself). Also, the fact that the Worklet is autonomous and its execution is asynchronous means that it requires less bandwidth than synchronous RPC.

The WVM, or Worklet Virtual Machine, is the execution environment and transportation mechanism for Worklets. The host adapter is the link between the WVM and the system component, which is essential for Worklet arrival, execution and dispatch. The WVM is a multithreaded system that can accept incoming Worklets from peer WVMs through either Java RMI or a direct socket connection.

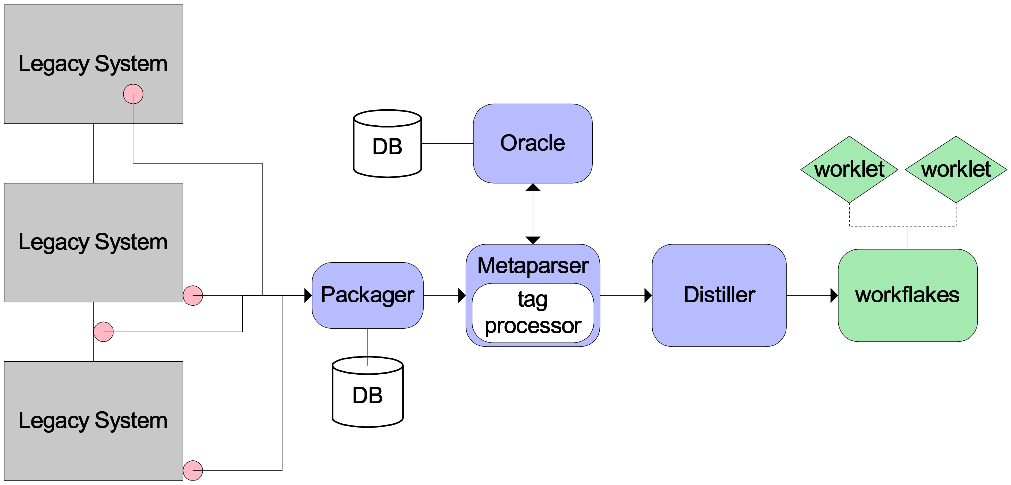
Incoming Worklets will have one Worklet junction intended for the local WVM, responsible for encapsulating the complete execution to be performed at that local WVM. There is also an optional Worklet jacket to specify control information for the Worklet junction. Such control information determines the overall behavior of the Worklet junction, such as the number of repeated executions, exit constraints, or initiation conditions. Worklet junctions tend to be newer than the target systems that they are looking to configure. This might necessitate automatic and dynamic bytecode download to allow for the assembly of the Worklet junction instance when it arrives at the target system.

Our solution involves a web server at every WVM, enabling it to serve up class bytecode for emitted Worklets. At the receiving end, a customized class loader downloads the bytecode of those classes that are not available locally from the most recent hops, or the origin WVM of the Worklet. To enhance survivability and improve efficiency, each WVM caches all transmitted and received bytecodes so that the bytecodes can be served up to any other WVM that might require it.

A further level of efficiency is achieved by using the Workgroup Cache [12] system to enable automatic bytecode-sharing among the WVMs along a Worklet's route. Since this mechanism allows WVMs to pre-fetch bytecode before the Worklet even arrives, it eliminates the need for explicit bytecode downloading at each WVM upon arrival of the Worklet.

# Applications and Examples

## KX



The KX event monitoring system is the primary application of our SmartEvents work. It is designed for the automated monitoring and reconfiguration of complex distributed systems. Probe data is processed by the Event Packager and Metaparser, and the resulting data is fed into the XML Universal Event System (XUES), an event-oriented rule engine. This looks for high-level and time-based situations. If XUES detects a condition that requires action, it issues a “high-level” event that is picked up by an Event Notifier (EN) component.

The EN determines the appropriate response and launches an appropriate handler, such as a Worklet-enhanced Cougaar [13] workflow ("Workflake"). The Workflake is capable of highly complex reconfiguration tasks.

## GeoWorlds

Our primary demonstration example was the GeoWorlds system from ISI [14]. GeoWorlds is an information management system integrating a digital library and a GIS. GeoWorlds makes extensive use of service components that may be either local or remote. The GeoWorlds' *SystemJobTable* data structure keeps track of all service requests. Failure of remote services is not handled gracefully in the current implementation.

Using the AIDE system from WPI, we instrumented GeoWorlds' *AbstractJobTable* class to send out a SmartEvent every time the job table was modified. The Metaparser extracts just the job ID and status information, and filters irrelevant status messages. XUES looks for remote events that appear to be hanging. If one is found, a Workflake is dispatched that notifies the user and uses a Worklet to clean up the job table.

## CHIME

The Columbia Hypermedia IMmersion Environment is a collaborative virtual environment being developed by PSL [15]. CHIME represents heterogeneous data in a homogenized "theme world". Using the 3D client, human users (avatars) can walk around the 3D world reflecting the infospace of interest, and interact with each other and with its data contents.

The CHIME server can dynamically import any available backend data source. For example, given a website URL, a component called FRAX (explained later) produces SmartEvents (FleXML-enriched Siena events) with metadata representing content from that data source. These SmartEvents are subscribed to by the Data Server (via Siena) and cataloged in an SQL database.

The Data Server then calls the Virtual Environment Modeler (VEM) to assign 3D objects to new elements in the database. The virtual world seen by 3D clients is then populated with the new data via the CHIME World Manager.

The File Recognize and XMLify component (FRAX) uses a URI to connect to a specified backend data source. FRAX recognizes the kinds of files or objects it is dealing with, and invokes the appropriate specialized ***plug*** written for that file/object type to extract metadata from the data source. The plug further instructs FRAX how to convert this rich metadata to an XML format (which is then packaged up into a SmartEvent).

ActEvents allow FRAX to dynamically accept new plugs with new semantics. We chose FleXML SmartEvents over Worklets because FRAX extracts and publishes structured metadata, fitting the XML-based model. This enables the CHIME Data Server to be able to dynamically handle new tags and associated content originated from new FRAX plugs.

Future work with respect to CHIME includes deployment of Worklets technology to allow more flexible KX-based monitoring of the CHIME subsystem. Such applications include 3D interpolation of backend changes via the use of KX-style gauges to analyze the actual change over time.

## InfiniTe

Dr. Kenneth Anderson and his group at the University of Colorado are developing an *"information integration environment"*, or InfiniTe, to aid software developers in performing complex information management tasks. In particular, they are focusing on supporting those tasks that involve *creatin*g, *findin*g, *maintainin*g, and *evolving* the relationships between software artifacts. Since InfiniTe is based on a model similar to FRAX for XML information exchange, SmartEvents are valuable for interaction between the InfiniTe's components and recipients. In addition, SmartEvents would aid in preserving relationships found between related artifacts.

### AI$^2$TV

AI$^2$TV [16] is a virtual collaborative environment for group study, distance learning, conference calls and video lectures in development under an NSF grant. A portion of the project involves a cache management system that controls the client's cache where video can be downloaded or prefetched. The entire system, including variables like bandwidth, is monitored by KX. Based on inputs from probes, KX can determine, for example, whether the client's cache controller should fetch a higher or lower compression quality stream from the Video Server.

Since AI$^2$TV uses CHIME as its virtual collaborative environment, ActEvents' benefits are inherited by AI$^2$TV. Further integration utilizing ActEvents is planned, including client scaling, proxy management and reconfiguration, and user management.

## Related Work

Many of the research topics described earlier in this paper have been examined in systems research, event infrastructures, active networks, and information management, besides those listed above under "Previous Work." We briefly discuss some of this related work here.

Our work on Worklets in ActEvent-based architectures closely parallels research done in active networks – "packets" serve as an abstraction for code-embedded ActEvents, and the idea of code injection is native to Worklets. Weatherall [17] abstracts away IP packets for a more generic code-embedded supertype, known as "capsules", and demonstrate how their model, using Java bytecode as their code mechanism, improves active network performance. Hicks, et. al. [18] instead define a restricted language for packet-based active networks. Such restricted languages are especially useful when performance during large volumes becomes a significant issue. While Worklets utilizes Java bytecode, any sufficiently flexible language can be used, given a set of WVM bindings for that language. Chin, et. al. [19] discuss utilizing active network technologies, including code injection, for connection rerouting. They discuss hierarchical topologies and dynamic connection rerouting; much of the same can be done, albeit at a higher level, with Worklets.

Event-based agents, like Worklets, are also used for increased autonomy. Das, et. al. [20] discuss an event description language, called MDL, to facilitate modeling by associating events with objects and attributes, to allow real-world scenarios. Such object- and attribute-enriched events also can be supported by ActEvents.

Rifkin and Kohare [21] summarize general event-based notification infrastructures. Utilizing such event-based architectures for debugging complex applications is not new. Bates [22] defines Event-Based Behavioral Abstraction (EBBA), which is concerned with developing models of complex, distributed systems, and simulating operation to speed debugging; ActEvents are similar, but often intended for run-time debugging. Page and Tufarolo [23] examine a particular case study of event-based Verification, Validation, and Accreditation (VV&A) under the auspices of the DoD, in both virtual (simulated) and run-time systems. Hilbert and Redmiles [24] utilize events within monitoring infrastructures, similar to the EBBA approach, for software engineering applications.

Event-based architectures can apply to more than just debugging systems. Sarin, et. al. [25] develop an event-based, object-oriented process model and system for generalized collaborative work around units of "work". Sateesh [26] discusses the application of time-bound event-driven models for real-time systems. Magee and Kramer [27] discuss event mechanisms within the framework of dynamic architecture description languages (ADL's). Finally, Cugola et. al. [28] discuss their JEDI event-based infrastructure for complex systems.

## Acknowledgements

We'd like to acknowledge John Salasin of DARPA for his help in developing the DASADA project and for numerous suggestions throughout this research. We'd also like to thank Ken Anderson, Antonio Carzaniga and Alexander Wolf of the University of Colorado for their advice for InfiniTe and Siena; Nathan Combs of BBN for help with Cougaar; Bob Neches, Ke-Thia Yao, and In-Young Ko of ISI for providing GeoWorlds as a useful target platform; David Garlan and Bradley Schmerl of CMU for their work on the Gauge API; and Bob Balzer of Teknowledge for work with run-time infrastructure standards

We thank George Heineman and Peter Gill for their work on the AIDE probe architecture. Finally, we'd also like to acknowledge PSL members who had a hand in this research, including Denis Abramov, Rose Alappat, Enrico Buonanno, Peter Davis, Joanna Gilberti, Shen Li, Kanan Naik, Michael Novich, Alpa Shah, Navdeep Tinna, Giuseppe Valetto, Simin Wang, and James Wu.

## References

---

1 Balzer, Robert. bbalzer@teknowledge.com. Private communication.

2 Carzaniga, A. and Wolf, A. Siena: Scalable Internet Event Noficiation Architectures. <http://www.cs.colorado.edu/~carzanig/siena> 2001.

3 Schmerl, Bradley. bschmerl@cs.cmu.edu. Private communication.

4 TIBCO Software Inc. "TIB/HAWK". <http://www.tibco.com/products/pdf/hawk_data.pdf> 2000.

5 Tivoli Systems Inc. "Tivoli Distributed Monitoring." <http://www.tivoli.com/products/documents/datasheets/distmon.pdf> 2000.

6 SunSoft Staff. "The ToolTalk Service: An Inter-Operability Solution." Prentice Hall PTR, 1993.

7 OMG. "Event Service Specification." <ftp://ftp.omg.org/pub/docs/formal/01-03-01.pdf> 2001.

8 Sun Microsystems Inc. "Java Messaging Specification 1.0.2b." <http://java.sun.com/products/jms/docs.html> 2001.

9 Luckham, David C. and Frasca, Brian. *Complex Event Processing in Distributed Systems.* Stanford University Technical Report CSL-TR-98-754, 1998.

10 Heineman, George. A Model for Designing Adaptable Software Components. 22nd Annual international Computer Science and Application Conference (COMPSAC-98). Pages 121-127, August 1998. Vienna, Austria.

11 G. Valetto, G. E. Kaiser, G. S. Kc, "A Mobile Agent Approach to Process-based Dynamic Adaptation of Complex Software Systems." *Proceedings of the 8th European Workshop on Software Process Technology (EWSPT-8)*, Written (Germany), June 19-21, 2001, Lecture Notes in Computer Science n. 2077, pp. 102 ff., Springer-Verlag, Heidelberg (Germany).

12 Kaiser, Gail, Christopher Vaill and Stephen Dossick. "A Workgroup Model for Smart Pushing and Pulling." 8th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, June 1999.

13 Combs, Nathan. ncombs@bbn.com. Private communication.

14 Neches, Robert, Ke-Thia Yao, and In-Young Ko. GeoWorlds: Integrated Digital Libraries and Geographic Information Systems. <http://www.isi.edu/geoworlds/publications.htm> May 2001.

15 Dossick, Stephen E. and Gail E. Kaiser. "Distributed Software Development with CHIME." ICSE-99 2nd Workshop on Software Engineering over the Internet, May 1999.

16 Kaiser, Gail and Giuseppe Valetto. "Ravages of Time: Synchronized Multimedia for Internet-Wide Process-Centered Software Engineering Environments". 3rd ICSE Workshop on Software Engineering over the Internet, June 2000.

17 Weatherall, David. "Active network vision and reality: lessons from a capsule-based system. *Proceedings of the 17th ACM symposium on Operating systems principles.* ACM, Charleston, 1999, pp. 64-79.

18 Hicks, Michael, et. al. "PLAN: a packet language for active networks." *Proceedings of the third ACM SIGPLAN international conference on Functional programming.* ACM, Baltimore, 1998, pp. 86-93.

19 Chin, Kwan Wu, et. al. "A model for enhancing connection rerouting using active networks." *Proceedings of the 2nd ACM international workshop on Modeling, analysis, and simulation of wireless and mobile systems.* ACM, Seattle, 1999, pp. 77-86.

20 Das, S., Caglayan, A. and Gonsalves, P. "Increasing Agent Autonomy in Dynamic Environments". *Proceedings of the second internal conference on Autonomous Agents*, Minneapolis, 1998, pp. 309-316.

21 Rifkin, Adam, and Kohare, Rohit. "The Evolution of Internet-Scale Event Notification Services: Past, Present and Future." KnowNow, Inc., 2000, http://www.knownow.com.

22 Bates, Peter. "Debugging heterogeneous distributed systems using event-based models of behavior." *Proceedings of the ACM SIGPLAN and SIGOPS Workshop on Parallel and distributed debugging.* ACM, Madison, 1988, pp. 11-22.

23 Page, Ernest H., et. al. "A case study of verification, validation, and accreditation for advanced distributed simulation." *ACM Transactions on Modeling and Computer Simulation, Vol. 7, Issue 3.* ACM, 1997, pp. 393-424.

24 Hilbert, David M. and Miles, David F. "An Approach to Large-Scale Collection of Usage Data Over The Internet." *Proceedings of the 1998 internal conference on Software Engineering*, IEEE, Kyoto, Japan, 1998, pp. 136-145.

25 Sarin, Sunil K., et. al. "A Process Model and System for Supporting Collaborative Work." *Conference proceedings on Organizational computing systems.* ACM, Atlanta, 1991, pp. 213-224.

26 Sateesh, T.K. "Conceptual Model of Real-Time Systems: A Perspective." *Proceedings of the 1995 ACM symposium on Applied computing*, ACM, Nashville, 1995, pp. 206-209.

27 Magee, J. and Kramer, J. "Dynamic Structure in Software Architectures." *Proceedings of the fourth ACM SIGSOFT symposium on Foundations of software engineering*, ACM, San Francisco, 1996, pp. 3-14.

28 Cugola, G., et. al. "Exploiting an event-based infrastructure to develop complex distributed systems." *Proceedings of the 1998 internal conference on Software engeineering.* IEEE, Kyoto, Japan, 1998, pp. 261-270.