

A Web-based Infrastructure for Awareness based on Events

Roberto S. Silva Filho, Max Slabyak, David F. Redmiles
Information and Computer Science, University of California, Irvine
Irvine, CA 92697-3430 USA
+1 949 824 4121
{rsilvafi, mslabyak, redmiles}@ics.uci.edu

ABSTRACT

The ability to be aware of other people's work in a collaborative environment is essential to improving the coordination of the members of a group. In this context, events originating from many sources have to be filtered and combined in order to provide the right information to the right person at the right time. As the Web becomes a popular and ubiquitous way to integrate different information sources, an infrastructure that allows the filtering, combination, abstraction and routing of awareness information is required. In this paper, we describe DEAL (Distributed Event Awareness Language), an event-based infrastructure and language that supports the development of awareness applications in the context of distributed collaborative environments. The requirements of the infrastructure are discussed as well as the language syntax. Our motivation was to design a language and a set of usable and useful strategies and web services that provide usable and useful awareness capabilities for the development of awareness applications.

Keywords

Event Notification, Distributed Awareness, Event Processing Languages, CSCW, Web Services.

INTRODUCTION

In a broader sense, awareness refers to people's ability to sense relevant changes in their environment. In the specific context of CSCW, awareness is usually related to the perception of the direct or indirect changes in the computational and social environments originated by other people in the group. These changes are usually communicated through different computational resources, devices and applications. Awareness is an essential element in distributed collaborative environments. Individuals and groups of people need to be aware of what other members of the group are doing, in special, the activities of these other

members that directly or indirectly affect each individual's work. For example, a manager needs to be aware of the progress of the tasks assigned to her subordinates, the presence of other members of the group in the workplace in order to see a demo, the arrival of a co-worker in a remote site, in order to start a chat session and so on. These activities can be modeled as events, atomic asynchronous messages that represent changes in the computational and social contexts.

The CSCW literature describes many ways to provide awareness in a collaborative and potentially distributed work environment [15]. Some examples include periodic pictures of a co-worker's office as provided by NYNEX Portholes [16], notifications about changes in shared artifacts, provided by the BSCW system [17], application monitoring gauges as described in [5] and so on. In order to integrate and combine the information coming from those different sources, allowing the processing, routing and filtering of such information, an event processing infrastructure is usually adopted [3].

Today's Web Services infrastructure focuses on providing a common set of protocols for the development of web applications, defining a web-based middleware for the development and integration of distributed web services [19]. In this infrastructure, the SOAP protocol is used as a remote procedure call mechanism for services communication; the UDDI implements a service location mechanism and the WSDL allows the description of the interfaces of the web services. This infrastructure alone, however, does not provide all the functionality necessary for the implementation of more sophisticated web applications [11]. In this context, event-processing services supplement this basic infrastructure providing the ability to integrate, process, select and route events originated from different nodes in the Web. In an event notification service, producers and consumers of information are separated by an event middleware that integrates these two parties, allowing different consumers to subscribe to information coming from many event producers. This event routing layer decouples event producers from their consumers, allowing the dynamic addition and removal of these components in the system.

*LEAVE BLANK THE LAST 2.5 cm (1") OF THE LEFT
COLUMN ON THE FIRST PAGE FOR THE COPY-
RIGHT NOTICE.*

Specifically, previous work analyzes the use of event notification servers in CSCW applications [4].

In this paper, we present DEAL (the Distributed Event Awareness Language), an event-notification infrastructure and language to support the development of distributed awareness applications for the Web. The DEAL environment extends the basic functionality provided by event notification servers such as Khronika [9], CASSIUS [8], CORBA Notification Service [12], ELVIN [6] and SIENA [1] to cope with the richer set of requirements of awareness applications. This is accomplished by the use of a powerful and usable event language that allows the definition, processing, combination, filtering and routing of events coming from heterogeneous sources (programs, applications, components, people, mobile devices and so on). The DEAL language syntax and resources were inspired in the features provided by event processing languages such as GEM [10], Yeast [2], EDEM [7] and READY [18]. Usability and simplicity with expressiveness were some of the principles considered in the design of the language. In special, the following scenario provides a set of requirements and motivations that guided the design of the language and infrastructure.

SCENARIO AND MOTIVATION

Consider an IT company with many branches in different cities over the country or even the world. Many people cooperate in different projects at the same time, performing different roles (manager, programmer, tester, designers and so on). Each project usually is carried on by a dynamic group of people whose interaction varies according to the group's current focus. At the beginning of a project, for example, designers and project managers are more active, whereas more towards the end, the activities concerning programmers, engineers and testers are more prominent. The work can also be split between different branches of the company. One branch, for example, deals with the product support while the other deals with the design and implementation. People join and leave groups as necessary. The group work is usually supported by different tools, such as configuration management repositories, word processing, CAD, databases and so on. These tools are used to produce and manage the evolution of the artifacts being developed, as well as their meta-information (documentation, specifications, metrics and so on). In this scenario, different people need to be aware of other group member's activities.

An indirect way of being aware of other people or group activities is to gauge the evolution of the artifacts they produce or modify. The kind of information one is interested in depends on her role in the organization. For example, programmers and engineers may want to be notified when some modules in a software project repository are ready for integration or when some change request is issued and consolidated in a defect report database. Managers, on the

other hand, can gauge the activity in certain project by visualizing a graph with the number of changes in the project repository over the day.

In this dynamic work environment, mobility and heterogeneity is another concern. Computers are no longer limited to users' desktops at their workplaces; instead they are increasingly mobile and ubiquitous. Users can interact with a collection of computational devices ranging from non-stop servers, workstations, and motion sensors to portable devices as laptops, mobile phones and PDAs. In this mobile environment, awareness applications have to adjust their intrusiveness and information delivery policies to comply with different contexts (time, place, physical, organizational, administrative roles so on). For example, managers may want to be constantly informed about the progress of their projects using their portable computers. The level of attention required by the user, however, is dependent on her context. For example, one may want to be immediately notified about a meeting when she is in her office. This same information, however, may not be very important when she is at home or on a business trip.

Information persistency is another important issue. A manager, coming from a business trip, for example, may want to gauge the progress of a project by analyzing the event history of the preceding week. This requires having a way to store events during a certain period of time so they can be delivered when the user's computer is on-line again.

Not all events, however, should be stored for further analysis. People usually do not want to be notified about transitory and ordinary events. For example, the arrival of someone in her office or the temporary unavailability of a printer, that ran out of paper. This requires a mechanism to discard old events and filter irrelevant information.

Finally, meaningful events are usually a result of or are expressed as a combination of more ordinary events whose occurrence usually obeys some predefined patterns. For example, the turning on of the light of someone's office followed by the typing of some characters in the computer keyboard may indicate the arrival of this person at her workplace.

REQUIREMENTS FOR AWARENESS APPLICATIONS

The previous scenario illustrated many features that our event-notification infrastructure must support such as the integration of heterogeneous event sources, the need to compose events, the ability to filter information, events expiration time and mobile applications support. Moreover, the appropriate delivery of an event depends on the user context, timing constraints, roles and priorities. The notion of groups is also required.

Functional Requirements

To cope with these requirements, the DEAL event language and infrastructure was defined to provide the following features.

Subscriptions: Logical expressions that provide the ability to select a subset of events based on their content or type. They allow the routing of events to the right person at the right time, with the appropriate priorities based on the user context, group, role and other properties.

Abstraction: A mechanism that combines different events into higher-level notifications in order to provide more meaningful awareness information. Event abstraction can use the following strategies:

- **Pattern matching:** The ability to subscribe to event sequences and patterns. It is the basis for abstraction and reduction. It may detect events in a specific order or out of order.
- **Reduction:** Translates sets of repetitive events, expressed as a pattern matching expression, into local state variables or higher-level events. This is specially required in monitoring applications, to prevent event flooding. A reduction consists in creating a new event indicating that an event pattern was detected.
- **Aggregation:** Is a more elaborated case of reduction in which the event generated is a composition of some of the attributes of the events in the detected pattern. Events are combined in a higher-level event which summarizes the content of the events in the pattern expression.

Event Condition Action (ECA) Rules: Special types of subscriptions that allow the execution of external applications or general actions whenever a logical condition is evaluated to true. For example: an action can add or remove subscriptions or even other rules; generate aggregated events, change policies, and invoke external applications in response to an event pattern detection. Rules can be used to evolve the behavior of the application in response to changes in the user environment.

Time constraints: Express Delivery intervals, time to live, as well as timing and temporal relations. Some events need to be detected within certain time interval in order to have some correlation. Transitory conditions may also be expressed by events with expiration time.

Subscription Priorities: Subscriptions are dependent on global, group and local contexts, allowing the adjustment of the information delivery according to the needs of the information consumers (or users).

Groups: Subscriptions and rules can be associated to groups, which are first class entities in DEAL language. This allows the broadcast of events, the definition of shared policies and contexts.

Hierarchical description of event sources: One of the neglected issues in some event infrastructures is the ability to answer the question “What can I subscribe to?” and “Which events are produced by each source?” The DEAL infrastructure provides the ability to browse through different event sources and to identify their events by keeping meta-information about which event sources are available and what events they produce.

Quality of Service Requirements

Apart from the main language features, the DEAL infrastructure allows the specification of different qualities of service and policies as follows.

Persistence of events and subscriptions: Events and subscriptions are persistent by default, allowing the support for mobile applications and pull delivery policy.

Mobility support: Clients are allowed to explicitly indicate their intent to move to a new location, allowing the infrastructure to perform the necessary migration operations such as the update event routing tables, the buffering of events or change the current qualities of service. This is performed by the **move-in/move-out** commands. Apart from these explicit commands, the infrastructure deals gracefully with the sudden disconnection of the event sources consumers.

Event Delivery Policies: During the specification of subscriptions and filters, one can specify which delivery policy to adopt, whether **pull** or **push**.

Security Policies: Authentication of groups, consumers and producers allow the event-processing infrastructure to prevent unauthenticated clients from receiving unauthorized events.

THE EVENT LANGUAGE

This section describes the DEAL event language. Examples are presented to illustrate its use and syntax.

Logical Expression Operators: `>`, `<`, `>=`, `<=`, `==` as well as `starts_with` and `ends_with`.

- `MyType:ev1.name starts_with "Ro"`

Subscriptions: Defined using the **subscribe** keyword, and removed by the **unsubscribe** command. Whenever a subscription is evaluated to true, a notification is produced having the list of events used in the subscription expression.

- `subscribe mySub SomeType:ev1.name == "Mike" and OtherType:ev2.counter == 2`
- `unsubscribe mySub`

Event Type Definition: Creates an event type, a structure supporting: boolean, string, long, int, as well as other Java basic types

- `type Type1 {name: string, age: int, is_present: boolean}`

On-the-fly event declaration and instantiation:

- `Type1:ev1 = {"john", 22, true}`

ECA Rules: Described by the use of the keyword **rule**, which uses the **do** command to define the action to be executed when the rule is matched.

- `rule myRule ev1.name == "check-in" and Local.time > 12pm do run myApplication.exe`

In this example, the **run** is a reserved word that allows the execution of external applications.

```
rule otherRule ev1.name == "turn-on-light"
and ev2.name = "workstation-activated" do {
type MyAbstrac = {name: String};
MyAbstrac ev3;
ev3.name = ev1.name + ev2.name;
notify ev3; }
```

Rule Activation: Activates and deactivates rules according to the context using the **enable** or **disable** commands.

- `rule activateRule1 Local.time > tomorrow do enable rule1`
- `rule deactRules ev1.description = "end of meeting" do disable rule1 rule2 rule3`

Temporal Expressions: Express time constraints between events. Time triggers and ranges: **at**, **by**, **in**, **within**.

- **at** 10pm - matches after the next occurrence of 10 pm
- **by** 10pm - matches from now until the next occurrence of 10 pm
- **in** 2 hours and 10 minutes - matches after 2 hours and 10 minutes from now
- **within** 3 hours - matches permanently in a period between now and 3 hours ahead

Time Period expressions: **today**, **daily**, **weekly**, **monthly**, **yearly**:

- **at** 10 am **daily**
- **at** **monday** **weekly**

Event Validity Check (time to "live"): Is a special attribute, present in all events, which expresses its expiration date and time. The expiration condition can be evaluated using the **expired** keyword:

- `expired ev1`
- `ev2.expiration < today and ev3.expiration < tomorrow`

Pattern Matching: Performs the matching of a sequence of events (repetition, sequence and optional).

- Enforced order: `ev1 then ev2 then ev3`
- Optional order: `ev1 and ev2 and ev3`
- Matching of repetition of events (0 or more and 1 or more): `repeat (ev1 and ev2) 2 times`

Groups: The keyword **group** allows the definition of sets of users, the keywords **add** and **remove** perform the addition and removal of users to a group, whereas the operand **in** checks the pertinence of users in groups.

- `group g1 { user1, user2, user3}`
- `add g1 user4 // adds user4 to group g1`
- `remove g1 user2`
- `ungroup g3 // removes the group`
- `user1 in g1`

Groups can be used as parameters of the **notify** command to broadcast events, as in the example

- `rule r1 ev1 then ev2 then ev3 do notify ev1 to g1`

Roles: A role is a group of users. Groups are used to represent roles. This allows users to perform different roles.

Contexts: Contexts are name spaces that define scopes where some properties, rules and subscriptions are valid. Local and global variables (or properties) can be stored in the local, group or global contexts. In addition, the contexts provide a set of predefined environment variables that allow the access to information as local time, host-name, user name and so on. The contexts are accessed through the special types **Local** and **Global**.

- `at 12 pm and Local.mycounter == 12`
- `at 12 pm and Global.members > 5`

Local and global rules can be defined. Expressions can use values of these contexts. A rule is associated to the local context scope using the **local** modifier. Global rules can be defined using the **global** modifier.

- `local rule myRule at 12 pm and mycounter == 12 do enable rule1`

In this example, *mycounter* is a variable in the Local scope. Each group has a special context associated with it. Group contexts are accessed by the group name, for example: `g1.size` expresses the size of the group.

Group rules can be defined using the **group** modifier followed by the group name, before the rule declaration.

Attributes can be added or removed from a context using the **addcontext** and **remcontext** keywords.

- `addcontext Local name:string`

Events and Subscription Priorities: Special attributes in the events, which are used by the system to perform event routing. They can be used in subscriptions by accessing the reserved attribute **priority**.

- `rule adjustPriority Local.time > 6 pm do ev1.priority = ev1.priority -1`

DESIGN

The DEAL architecture is described in Figure 1. The system is implemented as a wrapper around a notification server infrastructure such as CASSIUS [8] or Siena [1].

The event-processing kernel implements the DEAL functionality using the resources provided by the event notification server. Applications interact with the infrastructure through a programmatic API while end-users can use a command interpreter shell or a more sophisticated GUI. The interaction with the system can be intermediated by Web services interfaces such as SOAP or by lower-level protocols as HTTP/CGI. The architecture of the system can be distributed, using the resources of federated notification servers, according to the features provided by these systems. In special, Siena provides a scalable web-based content-routing infrastructure.

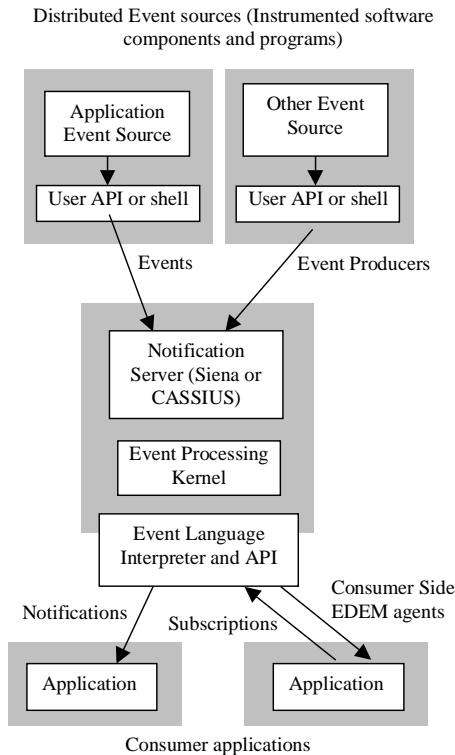


Figure 1 Design of the DEAL infrastructure using an event notification server

IMPLEMENTATION

DEAL is being implemented using the Java (J2SDK1.4) programming language and the CASSIUS notification server. CASSIUS was chosen for its ability to manage and provide access to a hierarchical list of event sources and their associated events. It also provides a subscription editor GUI that facilitates the interaction with the end users. For using the HTTP/CGI protocol, CASSIUS allows DEAL to be integrated with different event sources distributed over the Web, providing an event-based infrastructure for awareness information.

RELATED WORK

In this section, we summarize the main systems that inspired the DEAL language.

EDEM

EDEM (Expectation-Driven Event Monitoring) [7] is a user interface validation and monitoring tool. EDEM uses agents to monitor GUI event patterns according to design use expectations. The agent description language is very complete and allows the detection of event patterns, the manipulation of local context (in the monitored site), the definition of higher-level events (abstraction), the collection of repetitive events (reduction) and so on.

The EDEM architecture is defined in order to collect usability data. Since agents execute together with the application being monitored, the system was not designed to monitor events coming from multiple distributed applications.

Yeast

The Yeast (Yet another Event-Action Specification Tool) [2] is an event-action system used to automate tasks in a UNIX environment. Yeast allows actions to be performed when event patterns and environment changes are detected. It allows the association of temporal constraints to events, borrowing its syntax from the *at* and *cron* programs of UNIX systems. Sequential and out of order event pattern detection is supported. User-defined actions are executed whenever an event pattern match occurs. These actions can originate new events or start different applications. Yeast allows the definition of rules to be defined, activated or deactivated at runtime. This flexibility is provided by a shell script interface that integrates the UNIX shell commands with yeast pre-defined keywords.

The system is very complete, providing many features that can be used to support the development of awareness applications. It, however, was developed to operate on UNIX environments, being limited at monitoring its specific resources and objects such as processes, files and user logs. Users can define their own events but their types are not enforced. There is no advertisement of the event types provided by an event source. Event sources are not primary entities in this model. There is no explicit idea of subscription and subscriber. The event language does not allow the creation and manipulation of local variables, limiting the support for local and global contexts. User groups are not supported.

Khronika

The Khronika [9] is a centralized event notification service created to increase people awareness about their environment. One of the objectives of the system is to bridge the gap between computational and real-world events. Each user of the system can specify sets of pattern-action subscriptions that are used to automatically notify the user when an event pattern is detected. Khronika also allows the direct browsing of the events in the repository. Events have expiration time and remain on the server database as specified in their validity (days, hours or brief intervals). The event language allows queries by time interval, event

types and substring matching. Similar to Yeast, there is a mapping between English expressions as "today", "tomorrow", "now", "Thursday afternoon", and so on, to more precise time constraints. Access control lists and user groups are used. These restrictions are made simple for usability purposes.

Khronika does not provide the ability of abstracting and aggregating events. There is no support for different event sources, including mobile devices as well as the ability to activate/deactivate subscriptions (or rules) based on environment changes. There is no notion of user groups.

Gem

GEM [10] is a generalized event language for real-time distributed systems monitoring. It allows the event sequence detection and the specification of rules that can be activated or deactivated according to other rules. For being designed for real-time monitoring, rules can include special time constraints concerning incoming events delays. It also allows the use of event order constraints in event expressions, such as the specific order events should occur and the acceptable delay between them. Events can be abstracted and generated based on contents of other events. There is support for abstraction.

The GEM language itself was not defined for usability. It does not provide support for context and groups.

READY and CORBA

READY (Reliable Available Distributed Yeast) [18] is a general-purpose event notification service based on YEAST. READY adds to YEAST the ability to handle compound event matching, quality of service and other event constructs, in an implementation that extends the Standard CORBA notification server [12].

In its porting to CORBA [13], READY lost the simplicity, elegance and easy-to-use interface of the Yeast model. Its language became more complicated, being based on the OMG Event Notification Language. It also lost the timing constraints neutrality and elegance of Yeast.

CONCLUSIONS

DEAL is an event processing language and system designed to provide awareness information in a heterogeneous distributed system. The system was designed to cope with current distributed systems characteristics as mobility, heterogeneity, timing, as well as CSCW aspects as groups, context and priorities. In order to do so, it combines characteristics of different event processing, monitoring systems and awareness driven notification servers. It was specially designed as a distributed awareness service that can be integrated in many Web applications. This is accomplished by the use of the HTTP protocol. The event language was designed to be useful and usable, providing a high-level way to interact with the system. A prototype is being implemented in Java using CASSIUS as the basic notification service.

ACKNOWLEDGMENTS

This effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0599; by the National Aeronautics and Space Administration (NASA) under contract NAG2-1555; by the National Science Foundation under grants 0083099 and 0205724. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Laboratory, or the U.S. government.

REFERENCES

1. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332-383, Aug 2001.
2. B. Krishnamurthy and D. S. Rosenblum. Yeast: a general purpose event-action system. *IEEE Transactions on Software Engineering*, Vol. 21, No. 10. October 1995.
3. D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise System*. Pearson Education. ISBN 0-201-72789-7. Boston MA 2002.
4. D. Ramduny, A. Dix and T. Rodden. Exploring the design space for notification servers. *Proc. of the ACM CSCW'98*. pp. 227-235. Seattle, 1998
5. De Souza, C.R.B., Basaveswara, S. D., Kantor, M. Redmiles, D.F. Lessons Learned using Event Notification Servers to Support Awareness. *Human Computer Interaction Consortium '02- Winter Workshop*, Jan 31 to Feb 3, Fraser, CO. 2002
6. Fitzpatrick, G., Mansfield, T., et al. Augmenting the Workaday World with Elvin, *Proceedings of 6th European Conference on Computer Supported Cooperative Work (ECSCW 99)*, Kluwer, 1999, pp. 431-450.
7. Hilbert, D., Redmiles, D. An Approach to Large-Scale Collection of Application Usage Data Over the Internet, *Proceedings of the Twentieth International Conference on Software Engineering (ICSE '98, Kyoto, Japan)*, IEEE Computer Society Press, April 19-25, 1998, pp. 136-145.
8. Kantor, M., Redmiles, D. Creating an Infrastructure for Ubiquitous Awareness, *Eight IFIP TC 13 Conference on Human-Computer Interaction (INTERACT 2001 Tokyo, Japan)*, July 2001, pp. 431-438.
9. L. Lovstrand. Being selectively aware with the Khronika System. *Proc of ECSCW'91*. 1991.

10. M. Mansouri-Samani and M. Sloman. GEM: A Generalised Event Monitoring Language for Distributed Systems. In ICODP/ICDP'97. 1997.
11. M. Stal Web services: beyond component-based computing. CACM Special Issue: Developing and integrating enterprise components and services .Vol. 45, Issue 10. pp. 71-76. October 2002
12. Object Management Group. Notification Service Specification v1.0.1. August - 2002.
<http://cgi.omg.org/docs/formal/02-08-04.pdf>
13. OMG Common Object Request Broker Architecture – (CORBA/IIOP) v.3.0. formal/2002-06-33.
<http://cgi.omg.org/docs/formal/02-06-33.pdf>
14. P. C. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. ACM Transactions on Computer Systems. Vol 13. Issue 1. Feb 1995.
15. P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. ACM Proc. of Conference on CSCW'92. Toronto, Ontario, pp. 107-114. 1992
16. P. Dourish, and S. Bly. Portholes: Supporting awareness in a distributed work group, in Proceedings CHI'92, Monterey, CA, ACM/SIGCHI, 1992.
17. R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor and G. Woetzel. Basic Support for Cooperative Work on the World Wide Web. International Journal of Human Computer Studies 46, pp. 827-846, 1997.
18. R. E. Gruber, B. Krishnamurthy, and E. Panagos. High-level constructs in the READY event notification system. In 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications, Sintra, Portugal, September 1998.
19. Web Services Activity. 2002
<http://www.w3.org/2002/ws/>