

# Supporting Global Software Development with Event Notification Servers

Cleidson R. B. De Souza<sup>1,2</sup>

Santhoshi D. Basaveswara<sup>1</sup>

David F. Redmiles<sup>1</sup>

<sup>1</sup>Information and Computer Science  
University of California, Irvine  
Irvine, CA, USA

<sup>2</sup>Department of Informatics  
Federal University of Pará  
Belém, PA, Brazil

{cdesouza,sdb,redmiles}@ics.uci.edu

crbs@ufpa.br

## ABSTRACT

Along with the rapid globalization of companies, the globalization of software development has become a reality. Many software projects are now distributed in diverse sites across the globe. The distance between these sites creates several problems that did not exist for previously co-located teams. Problems with the coordination of the activities, as well as with the communication between team members emerge. This paper describes how event notification servers are a useful technology for supporting global software development. They can facilitate the development of collaborative tools, which can be used to support distributed software development. In general, advantages of using event notification servers are described as well as specific problems that they help solve.

## Keywords

Global Software Development, Distributed Software Development, Cooperative Software Development, Event Notification Servers.

## 1. INTRODUCTION

Nowadays, globalization is a word that has been heavily explored. Basically, it means that the economical, cultural and social boundaries of countries are disappearing. For example, in the United States you can buy the same products that you can find in Brazil. Technology innovations are ubiquitous. The roaming feature of cellular phones allows people to travel and keep in touch with friends and coworkers. Cultural barriers are also becoming subtler. In Brazil, for example, you can watch the same television series that are being presented in the US. Software is no exception to this rule. For example, Microsoft derived 55% of its sales from outside the US [19]. Due to different government regulations, several companies, especially in telecommunications, need development sites in different countries to get a position in the local market [16].

Therefore, these companies need to adapt their processes, tools, and organizational culture to overcome the disparity between the sites. In order to do so, they need to solve a wide variety of problems. The most obvious is the physical distance between the sites [13]. In this case, the sense of working in a team decreases due to the lack of interaction between the members of different sites. As a consequence, there is a lack of trust because the members usually do not have knowledge about overseas' culture. Relatively simple activities like discussing requirements in meetings can not be performed. There are also strategic, cultural, knowledge management, project (and process) management and technical issues to be solved [15]. All these problems must be understood and properly resolved for global software teams succeed.

Our position in this paper is that event notification servers can be used as an infrastructure to support global software development. Several technical problems that arise can also be solved. For example, using these servers, the lack of informal communication can also be partially solved by building collaborative tools, such as instant messenger systems. In general, notification servers can be used as a framework on which collaborative tools can be built to improve global software development. Indeed, several popular applications such as instant messenger systems, information tracking [3], application gauges [9], and workflow systems [7] have been built with notification servers.

In order to be able to use event notification servers, events (representing activities) must be captured. One way to capture these events is through instrumentation of applications. Another way is through event monitoring [17]. Others, such as MILOS [21], use a publisher-subscriber component in its architecture. In this short paper we are not focusing on how these events are captured, but rather are assuming that events are captured and therefore supplied to the notification server.

The rest of the paper is organized as follows. A short description of event notification servers is presented. Then, the next section presents the advantages that can be obtained using notification servers. Finally, some conclusions and future work are described.

## 2. EVENT NOTIFICATION SERVERS

There is a growing trend in creating distributed software applications that run over the Internet. Since the Internet is not

reliable, a good way to think about how these applications work is as loosely coupled autonomous objects. Instead of using direct communication, these applications use an event-notification or publish-subscribe design style. Messages are not sent from an application directly to another, they are sent to an event dispatcher, or notification server, which in turn distributes them to the applications that need them. In this case, there are two types of components: information sources and consumers. The information sources publish events, while the information consumers subscribe to particular categories of events. Events are sent by the information sources to the notification server, which ensures the delivery of these events to all interested information consumers. In general, in an event-based system, component interactions are modeled as events, which are transmitted in the form of notifications [25].

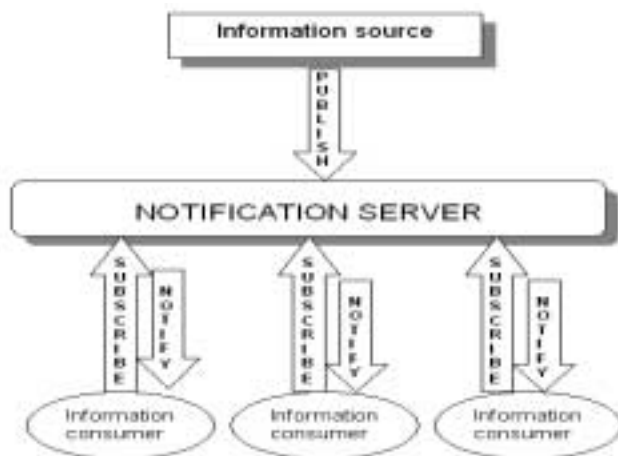


Figure 1. A generic event notification server.

A notification server usually provides a *subscription service* that allows consumers to subscribe to different types of events using, for example, boolean (logical) connectors, regular expressions, sequence matching, and so on. There are many event notification servers in existence and they are surveyed elsewhere [7]. Three representative examples are CASSIUS [18], Elvin [11] and Siena [4]. CASSIUS was developed tailored to provide awareness information for groups. Elvin was originally developed to gather events for visualizing distributed systems, but it evolved later to a multi-purpose event notification system. Finally, Siena emphasizes scalability in distributed environments.

### 3. ADVANTAGES OF EVENT NOTIFICATION SERVERS

Herbsleb and Moitra [15] classify the dimensions of the problems encountered in global software development: strategic issues, cultural issues, inadequate communication, knowledge management, project and process management issues, and technical issues. Each one of these problems demands different approaches. In this paper, we argue that event notification servers can be used to support global software development because they help resolve the technical issues, they help to minimize the inadequate communication between developers, and they provide support for some organization issues such as role support.

### 3.1 Flexibility

Event notification servers can provide several advantages when used to support distributed software development. The most important is the decoupling between the sources and consumers of the events. When coupling is decreased, the flexibility of the entire system is improved. For instance, an information source publishes their events without knowing which components are interested in them. Meanwhile, the information consumers subscribe to the events in order to be notified without knowing which component is the source of the event. In this manner, if a new information producer is added to the system, no modification is necessary to it. Furthermore, if a new information consumer wants to be notified about an event, it simply needs to subscribe to it. The information source does and the other components of the system do not need to be modified. In short, the event technology enables a “plug and play” approach to support introduction of new service components [7].

Since wide-area networks are often slow and unreliable, this flexibility is important because it might help to avoid problems in one site because of other sites since there is no strict dependency between them. It also easily supports the addition of new components, therefore allowing the interconnection of new sites and their tools.

### 3.2 Integration

An event notification server can also be used to support integration among software development tools [22]. Therefore, these tools will be able to communicate with each other and cooperate despite incompatible data formats. In fact, event based integration is one of the most prevalent integration approaches to support *loose* integration [1]. A *loose* integration approach occurs when tools have little or no knowledge of one another reducing the coupling between tools. This advantage is particularly important in distributed software development because, often, each site uses different tools as they became part of the company due to acquisitions and merges [16], i.e. the event-based framework can be used as a mechanism to integrate tools and therefore support cooperation amongst distributed sites. Frequently, sites also have their own organizational culture that encompasses the unit’s norms and values, and also the culture of system development, such as the use of methodologies and project management practices.

### 3.3 Internet-scale Notifications

Another important advantage is the possibility of Internet-scale event notification. Although this feature is not implemented in all notification servers it has becoming increasingly more common. Siena [4] is an example of a server supporting wide-area event notification. Basically, this means that those notification servers enable the construction of Internet-scale distributed applications [3]. This feature is necessary to support integration among tools, since these tools are used in distributed sites. Furthermore, it is also useful because it enables the communication and exchange of information (through the exchange of events) among the distributed different sites.

### 3.4 Support for Informal Communication

The physical distance between sites makes it harder for distributed team members to locate remote colleagues and to be

aware of their actions. This lack of awareness implies in fewer interactions since they can not tell what people are up to and if it is appropriate to interrupt [2]. In fact, one of the main problems in distributed software development is the lack of informal communication [14], i.e., since the teams are not physically together, they can not meet around the water cooler, in the hallway or in other public areas. Empirical studies suggest that informal communication is very important to coordinate teams in uncertain tasks such as software development [8][20]. In other words, software developers rely also on informal, ad-hoc communication to fill in details, handle exceptions, correct mistakes and bad predictions, and manage the effects of all these changes [14]. As this type of communication is absent in global software development the coordination of these tasks is much more difficult.

Portholes systems were one approach to give geographically distant co-workers a sense of mutual awareness [10]. More recently, researchers have tried displacing porthole's information in time to compensate for differences in time zones. Another possible approach to this problem is to use Instant Messaging (IM) to stimulate informal communication (opportunistic interactions) among workers at different sites. Usually, IM systems provide awareness information about the presence of others. For example, if the other members of the team are connected at the moment, the last time they connected and so on. Notification servers have been used as infrastructure to build IM systems [3], in so doing they can indirectly support informal communication.

### 3.5 Role Support

In general, software development is essentially a cooperative activity performed by a team. This is also true for distributed software development where developers are dispersed in different sites and even countries. Roles are used to help the coordination of a large number of developers. Examples of roles are: project managers, team leaders, senior analysts, designers, programmers and so on. Each role has its own duties: e.g., the project manager is responsible to ensure that the project is completed within budget and on time, while the reviewer has to ensure that the components being implemented do not have defects. These formal roles need different types of information to be properly played. In fact, according to [24] a collaborative system should be able to direct particular information to particular people based on their roles.

Fuchs [12] argues that the same event can be highly important for one user and completely uninteresting for another user in the same situation, or similarly, an event may be important for a user in some situation and irrelevant for the same user in another. Fuchs' work is about public administration, but the similarities to software projects is sufficient that we believe similar conclusions apply.

In general, in an event-based system there is communication: between the information source and the notification server and between the notification server and the information consumers. There are two types of architectures to implement this communication: pull and push. In a pull architecture, the component interested in receiving the events is responsible for contacting the other component to check for new events. Usually, it is necessary to specify how often one component wants to poll the other. For example, CASSIUS uses a pull

architecture to communicate with the information consumers, i.e. they poll the notification server to check for new events, in our scenario, at intervals of five seconds.

On the other hand, in a push architecture the component that wants to send the events "pushes" these events onto the others. In other words, the second component is "called" by the one that is sending events. In this case, these components need to implement some kind of standard interface to be invoked by the other components.

To the best of our knowledge, current notification servers implement one of the two architectural models. However, in order to support distributed software development both architectural models should be supported due to the different roles involved in a software development activity. For example, a push model is important in order to inform the manager of an invalid or problematic state. In real-time scenarios, it might be imperative to receive a notification of events as soon as they occur. On the other hand, a pull model is important in a mobile environment where the information sources can be disconnected. In this case, the events would be lost if they were not recorded for later delivery.

### 3.6 Network Problems

In order to support global software development, networks spanning across several sites are necessary. However, these networks are often slow and unreliable, i.e., network packages are lost due to problems in the transmission. For several applications, this is not a problem, however in global software development the events flowing in the network encapsulate critical data necessary to improve coordination of activities, communication, etc.

Event meta-information could be used to avoid networks problems. By meta-information, we mean any additional information about an event that is recorded by the information source or notification server and sent alongside with the event. Therefore, this meta-information is available to the information consumers, in addition to event information. Typical examples of meta-information are the type of the event being sent, the information source that sent the event, the timestamp when the event was sent by the information source and so on. A flexible mechanism to support event meta-information could be used to support additional services such as guaranteed delivery and security, therefore alleviating the problems of the wide-area networks.

## 4. EXAMPLE SCENARIO

The following scenario illustrates how a notification server might be used to support distributed software development. It is similar to the scenario developed by [23] while describing Palantír. Palantír's goal is to support developers with advanced graphical views that keep them continuously aware not only of which artifacts are changing, but also of the impact and severity of those changes. We are currently working with the developers of Palantír to implement the following scenario.

In this scenario, several distributed users are working on different, but dependent, parts of the system. For example, John is developing a component called "Note.java" in Los Angeles,

USA. This component depends on another component “Main.java” being developed by another developer, Simon, in Kerala, India. The dependency between these components implies that John needs to be notified when component “Main.java” is modified. Otherwise, changes in Main.java may “break” his code. In other words, John adds this component to his private workspace in Palantír; therefore this system will send a subscription to a notification server specifying that John be notified about changes (events) for that component. This subscription is sent to the notification server located in Los Angeles, which forwards it to all other notification servers of the company including one in India. Meanwhile, Simon, who has been working in India to complete his task, checks in “Main.java” into the Configuration Management (CM) system.

The CM system is instrumented in a manner such that it sends events to the notification server, when components are checked in or checked out. In this case, a new event viz. “*component Main.java checked in by Simon at 5:03 PM*” would be generated. The notification server in India adds information about the current location to the event and then forwards it to the server in LA. The next morning, John’s workspace would have an indication of changes to the component “Main.java”. Using Palantír [23] John would be able to figure out the appropriate action to be taken: change his code, contact Simon, etc.

Similarly, say John’s component also depends on another component, “Connection.java,” being developed by Susan in New York, USA. In this case, due to the small difference between the two time zones, John could be working on his component at the same time that Susan is working on hers. In this case, John would be notified about changes in this component accordingly and he might contact her to avoid possible problems and inconsistencies.

There are several important points in this. First, the notification server must support Internet-scale notifications in order to deliver the information among the distributed sites. Second, the notification server must support push and pull. A pull architecture is necessary to store events and deliver them when necessary as in the first part of the scenario. Push is used in the second case to deliver information in near real time. Third, in this scenario, events are being treated as strings without any formal structure. However, in reality, events should be semi-formally structured in order to (1) facilitate their processing, and (2) provide support for error detection. It is also necessary to have some mechanism for defining event structure and type. For example, events being sent by a configuration management system are different for those provided by project support tools such as MILOS [21]. Finally, it is important to mention that, to the best of our knowledge, there is no current implementation of a notification server that supports all the features described above. We are currently building a more complete server and plan to deploy it in a real organization with other collaborative software engineering tools in order to investigate whether the advantages described in this paper can be obtained.

## 5. CONCLUSION AND FUTURE WORK

Software development performed by traditional co-located teams is an intrinsically difficult task. It is difficult because of the complex nature of the activities being executed requiring

strong coordination and communication among developers. When performed in a distributed setting, software development becomes even more challenging. In global software development, several problems arise due to the physical, social and cultural difference between the developers. For example, communication is much more difficult because of language barriers, informal communication among the team members cannot happen and trust is more difficult to achieve.

The goal of this paper was to present event notification servers as an important and useful technology to support global software development. They are able to do so because they provide a good framework to the development of collaborative tools. For examples, technical problems that collaborative tools face in widely distributed scenarios can be solved using these servers. Indeed, we presented some common problems in global software development and how event notification servers could solve them.

Finally, it is important to note that event notification servers are one mechanism to distribute events across the sites. They provide the infrastructure for a variety of tools to be built upon, such as process visualization [6] or event reasoning [5].

## 6. ACKNOWLEDGMENTS

Effort 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. Effort also partially funded by the National Science Foundation under grant number CCR-0205724 and 9624846. 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 the Defense Advanced Research Projects Agency (DARPA), the Air Force Laboratory, or the U.S. Government. The first author was also supported by CAPES (grant BEX 1312/99-5).

## 7. REFERENCES

- [1] Barret, D. J., Clarke, L. A., Tarr, P. L. *et al.* A Framework for Event-Based Software Integration, *ACM Transaction on Software Engineering*, vol. 5, n. 4, pp. 378-421, October 1996.
- [2] Bellotti, V. and Bly, S. Walking away from the desktop computer: distributed collaboration and mobility in a product design team. *In Proceedings of the ACM Conference in Computer Supported Cooperative Work*, pp. 209-218, Cambridge, MA, ACM Press, 1996.
- [3] Cabrera, L. F., Jones, M. B. and Theimer, M. Herald: Achieving a Global Event Notification Service, *In Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Germany. IEEE Computer Society, May 2001.
- [4] Carzaniga, A., Rosenblum, D., and Wolf, A.. Design and Evaluation of a Wide-Area Notification Service. *ACM Transactions on Computer Systems*, 19(3), 332-383, 2001.
- [5] Cohen, D., Feather, M. S. and K. Narayanaswamy. An

- Event-Based, Reactive, and Extensible Infrastructure for Distributed Collaboration, In Proceedings of the *Workshop on Coordinating Distributed Software Development Projects*, IEEE Intl. Workshops on Enabling Infrastructure for Collaborative Enterprises, June 1998.
- [6] Cook, J. Internet-based Software Engineering Enables and Requires Event-Based Management Tools, *Proceedings of the 3rd Workshop on Software Engineering over the Internet*, At International Conference on Software Engineering, Limerick, Ireland, June 2000.
- [7] Cugola, G.; Di Nitto, E.; Fuggetta, A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9), pp. 827–850, Sept. 2001.
- [8] Curtis, B., Krasner, H. and Iscoe, N. A Field Study of the Software Design Process for Large Systems. *Communications of the ACM* 31(11):1268-1287, November 1988.
- [9] De Souza, C. R. B., Basaveswara, S. D., Redmiles, D. F. Lessons Learned Using with Notification Servers To Support Application Awareness, In progress.
- [10] Dourish, P. and Bly, S. Portholes: Supporting Awareness in a Distributed Work Group. In Proceedings of the *ACM Conference on Human Factors in Computing Systems*, Monterey, CA, pp. 541-547, 1992.
- [11] Fitzpatrick, G., Mansfield, T. et al. Augmenting the workaday world with Elvin, In *Proceedings of 6th European Conference on Computer Supported Cooperative Work*, pp. 431-450. Copenhagen, Denmark, Kluwer, September 12-16, 1999.
- [12] Fuchs, L.. AREA: A cross-application notification service for groupware. In *Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work*, pp. 61–80, Copenhagen, Denmark. Kluwer, September 12-16, 1999.
- [13] Grudin, J. CSCW: History and Focus. *IEEE Computer* 27(5):19-27, May 1994.
- [14] Herbsleb, J. and Grinter, R. Architectures, Coordination, and Distance: Conway's Law and Beyond, *IEEE Software*, September/October, 1999.
- [15] Herbsleb, J. and Moitra, D. *Global Software Development*, IEEE Software, vol. 18, Issue 2, pp. 16-20, March/April 2001, Special Issue in Global Software Development.
- [16] Herbsleb, J., Mockus, A. Finholt, T. A. and Grinter, R. *Distance, Dependencies, and Delay in a Global Collaboration*, In Proceedings of ACM Conference in Computer Supported Cooperative Work, pp. 319-328, Philadelphia, Pennsylvania, 2000.
- [17] Hilbert, D., Redmiles, D. Extracting Usability Information from User Interface Events. *ACM Computing Surveys*, accepted and to appear, Vol. 32, No. 4, December 2000.
- [18] Kantor, M., Redmiles, D. Creating an Infrastructure for Ubiquitous Awareness, Eight IFIP TC 13 Conference on Human-Computer Interaction (INTERACT 2001), Tokyo, Japan, pp. 431-438, July 2001.
- [19] Karolak, D. W. *Global Software Development*. Chapter 1, "What's Driving Global Development?", IEEE Press, 1999.
- [20] Kraut, R. E., Streeter, L. A., Coordination in software development, *Communications of the ACM*, 38(3), pp.69-81, March 1995
- [21] Maurer, F., Dellen, B., Bendeck, F., Goldmann, S., Holz, H., Kotting, B., and Schaaf, M. Merging Project Planning and Web-Enabled Dynamic Workflow Technologies, *IEEE Internet Computing*, pp. 65-74, May/June 2000.
- [22] Reiss, S. P. Connecting Tools Using Message Passing in the Field Environment, *IEEE Software*, pp. 57-66, July, 1990.
- [23] Sarma, A. van der Hoek, A., Palantir: Increasing Awareness in Distributed Software Development, *In Proceedings of the International Workshop in Global Software Development*, At International Conference on Software Engineering, Orlando, Florida, May 2002.
- [24] Steinfield, C., Jang, C.Y., and Pfaff, B. Supporting Virtual Team Collaboration: The TeamSCOPE System. In *Proceedings of ACM GROUP Conference*, pages 81–90, Stockholm, Sweden, Nov. 1999.
- [25] Sutton, P., Arkins, R., Segall, B. Supporting Disconnectedness – Transparent Information Delivery for Mobile and Invisible Computing. In Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), 2001.