A Script-based Approach to Distributed Presence Aggregation

Olaf Bergmann[†] Jörg Ott[‡] Dirk Kutscher[†]
[†]Universität Bremen, Technologie-Zentrum Informatik, E-mail: {bergmann, dku}@tzi.org
[‡]Helsinki University of Technology, Networking Laboratory E-mail: jo@netlab.hut.fi

Abstract

Personal presence systems are widely used to get aware of other users' availability and willingness to communicate before actually contacting them. After early systems focused on ad-hoc text messaging only and relied on manual updates of status descriptions, modern applications not only integrate multi-media communication but also facilitate automatic detection of status changes. The status of devices owned by a particular user then can be used to infer the user's presence status automatically, i.e. without explicit user-interaction. To achieve this, unstructured data provided from various sources is aggregated and set into relation with user-specific context information. The aggregation service presented here eliminates the need for extensive information acquisition that is necessary for most learning algorithms. Users instead have full control over the aggregation process, including the distribution of their presence status information to interested watchers.

keywords: Presence aggregation, sensor fusion, capability description, SIP, PIDF

1. Introduction

Interpersonal communication is an important aspect of enterprise organization and thus can have significant impact on the employees' productivity if not handled efficiently. Recently, instant messaging and presence systems have been recognized to break the predominance of email as the most important means of informal asynchronous communication. Some systems even provide support for automatic detection of users' presence status, hence making manual updates of status descriptions redundant. To do so, low-level sensors like motion detectors and personal communication devices are used to infer the user's physical presence at a specific location and whether or not the user is busy.

The availability information then can be published to anyone who has indicated interest and who is authorized to obtain this information. To facilitate efficient and secure distribution a generic event notification protocol can be used, e.g. the Session Initiation Protocol (SIP) [15] extension defined in [11]. Standardized syntax and semantics for presence information exchange are specified in [10].

In SIP, user mobility is enabled by soft-state registrations of contact addresses at a specific registrar server that is usually operated by the administrative owner of the user's domain. Hence, registrations for `john@example.com` would somehow arrive at a server that accepts SIP requests for the domain `example.com`. Whenever a SIP request is sent to `john@example.com`, it is directed to the user's current contact address by that particular server. As multiple contacts can be stored for a particular user the relative priority of every address can be specified explicitly.

Even though the priority mechanism comprises a simple form of presence-based request routing, registrations neither provide rich presence information nor do they reflect rapid changes of the user's availability—especially when moving around in wireless networks. To overcome this limitation, the SIP proxy can be co-located with a presence agent (PA) that aggregates the presence notifications of the user's personal devices. As a result, the PA can use the synthesized information to make enhanced routing decisions, and subscribed users may be notified of the aggregated status information.

Presence aggregation therefore happens at different abstraction levels and at different points within a network: End systems often use *sensor fusion* to infer a user's presence status. With this approach, low-level sensor data is examined in a closed environment—e.g. the user's office—representing a distinct *presence zone*. After that initial aggregation step, the status descriptions for a presence zone are published via SIP to a PA that manages subscriptions for the corresponding user. Having various communication devices in several presence zones, the user's PA has to merge the presence descriptions that are published from multiple zones. At this point, the presence information is represented as XML document that is not suited well as input for the sensor fusion approach. Instead, we propose script-based aggregation of presence documents from multiple sources that can be individually controlled for every subscribed watcher. An extension to the standardized presence information data format facilitates secure publication of watcher-specific views on the user's presence status.

This paper is structured as follows: Section 2 references existing research activities with relevance to our work. The architecture we are using to convey presence information is shown in section 3, followed by a detailed discussion of script-based aggregation in section 4. Our experience is reviewed and future steps are outlined in section 5. Finally, section 6 gives a brief summary of our achievements.

2. Related Work

Numerous research activities in the past have addressed services that transparently collect context information in order to support mobile applications, particularly to foster efficient communication [17], [2], [6]. The context information being used in most cases is made up from sensor data that has been aggregated and transformed in appropriate representations. For instance, to facilitate determining user location, several sensors could be related, like wireless signal strength, GPS, direction, velocity and possibly weather conditions.

The mechanism used for aggregation and transformation depends on the input data types and the desired output type. While, e.g., coordinate triangulation requires floating point operations yielding a geographic location as result, estimations on the availability of a human user to receive calls on her mobile phone merely use symbolic values like *away* or *busy*. In addition to manually crafted functions, sensor fusion can be automated using AI technologies like learning algorithms and fuzzy technology.

2.1. Personal Presence Systems

Since the early days of the Internet, networked applications have been used to leverage communication between people at distant locations. [4] credits the UNIX commands `talk` and `write` as the roots of *instant messaging*, i.e. text-based ad-hoc communication in a networked environment. Besides, the *finger* protocol [19] has been used for a long time to retrieve user-specific status information from remote hosts.

An important aspect of instant messaging systems is the awareness of other users, i.e. the concept of *personal presence*. One of the key features of messaging applications like AIM or Windows Messenger is the ability to display symbolic status descriptions for remote users' contact addresses. A limitation of these systems is the lack of support for multiple communication channels. A subscribed user is represented as a single contact with its aggregated presence status. A calling user hence cannot select explicitly which channel should be used for communication. Some call signaling protocols instead provide a mechanism to negotiate which communication facility is to be used, thus preventing the explicit decision of the calling user which is a fundamental part of interpersonal communication [8].

2.2. Generic Event Notification Systems

Being a popular research topic for more than a decade, numerous publications are available on Internet-scale event notification systems (ENS), including dynamic aggregation of event notifications.

[1] defines an abstract framework for generic event notification systems in wide-area networks, based on previous work from Rosenblum and Wolf (see [16]). The notification service is formally described by a number of operations to be invoked on the system, where event notifications are treated as sets of typed attributes. *Filters* on attributes can be used to decide if an event matches a specific subscription or advertisement. Moreover, filters can be combined to *event patterns* that allow selective forwarding of notifications. The architecture thus facilitates filtering of events but does not modify notifications.

A similar approach to graph-based forwarding of event notifications is documented in [5] for the *Solar* system. Using a central control component, the dissemination of event notifications is modeled as a directed acyclic graph. Any node in this graph may aggregate several event streams into a single output stream. To receive events, a node previously must have subscribed to other nodes upstream.

The Solar system provides an overlay network of event brokers that can be used by applications to subscribe to relevant context information synthesized from sensor data that is collected at the graph's leaves. This architecture is an example for a plethora of middleware layers currently emerging that aim at "ubiquitous" or "pervasive" computing.

2.3. IETF Standardization

Standardization of architectures and protocols for interoperable presence services has been in the focus of the IETF for several years. In addition to defining subscription-based notification services—SIP being only one of them—considerable effort has been taken to guarantee interoperability between these services. As a result, a data format called PIDF [18] has been developed to represent a basic information set for presence, together with a set of application rules (CPP, [10]). XML notation and an extension mechanism for the core format have fostered several special-purpose extensions, e.g. to include geographic location information or to express additional descriptions of the publishing entity.

When user presence gained importance and the CPP turned out to be an insufficient description of a presence service's semantics, a formal data model for user presence within the IETF was defined [12]. In that document, the aspects of service capabilities and device status have been separated for the first time from the presence status of an individual user. A vocabulary for capability descriptions currently is under development [7].

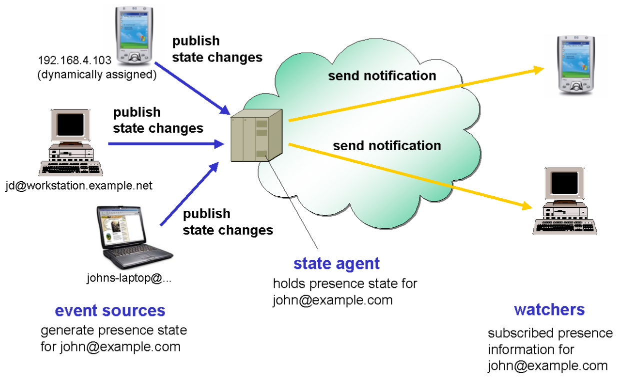

Figure 1: SIP service architecture for event notifications.

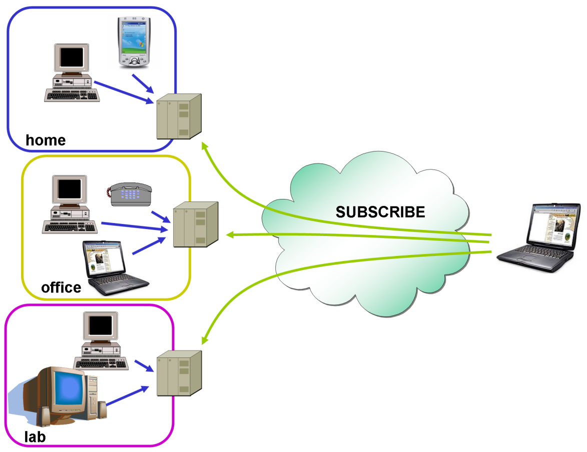

Figure 2: Subscriptions to multiple presence sources.

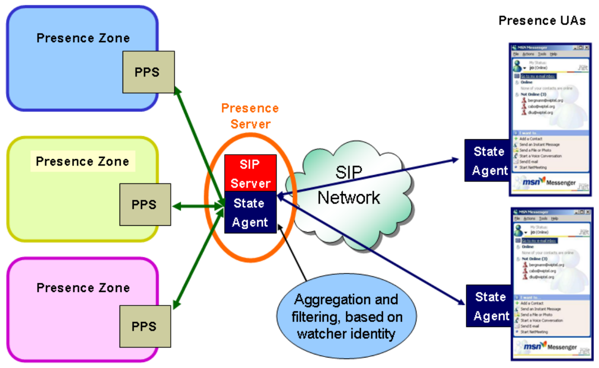

Figure 3: Aggregation of presence notifications.

Targeting at large-scale data distribution, SIP provides a scalable architecture for secure exchange of small presence documents given that updates are not too frequent (say, every 120 seconds). To combine SIP services with automatic status updates based on low-level sensor data, an additional aggregation step is necessary to avoid overloading the SIP presence agent (PA). When dealing with multiple presence zones for a particular user, the PA must be able to merge status descriptions from various sources. However, users should be given fine-grained control over the aggregation process, including the contents of notifications that are sent to subscribed watchers. In the following section, we give an overview of our SIP-based architecture that uses a hierarchy of aggregation servers to achieve this without downgrading overall performance or security.

3. System Architecture

As mentioned before, we are building upon the SIP event notification service as described in [11]. Entities that are interested in the current status of a particular user may subscribe the user's status identified by a generic presence URI and get notified whenever a change occurs. Figure 1 depicts the common SIP service architecture for basic event subscriptions. On the left, publishing entities send status update messages to a server that manages subscriptions and notification generation on behalf of the entities within that particular presence zone. The subscribed entities on the right then will be notified of any status change as soon as it has been signaled to the presence server located in the middle of this image.

3.1. Aggregating Multiple Presence Sources

Now, consider multiple sources contributing to the status description of a mobile user, as shown in figure 2. Every rectangle on the left hand side of the image depicts a self-contained presence zone, with consistent knowledge of the status of every user within that zone. As a user can be part of several disjoint zones, inconsistent status information may occur when subscribing to multiple zones as is the case in most multi-protocol instant messaging systems, shown at the image's right.

Adding a centralized entity that has sufficient knowledge about the distinct zones the user is part of allows to aggregate the streams of independent status notifications to form a consistent view on the user's presence status, as shown in figure 3.

Here, the presence server located in the middle aggregates the information from contributing sources at the left. The server acts on behalf of the user whose presence information is to be published. In particular, the server must be authorized to handle subscriptions and generate status notifications for that user—bypassing the presence agents at the edge of the zones where the original presence documents have been generated.

Note that the centralization of aggregation servers refers only to the administrative domain of a particular user (i.e., there is no global root server for presence aggregation, and the DNS is being used to determine the server that handles requests for the user). As with HTTP or SIP for voice communication, every domain can provide a server farm to do the processing for that particular domain. Load sharing and replication of servers then assure high availability and considerable performance of the presence service.

The aggregation process is controlled by user-specific rule-sets that specify how concurring presence notifications from the contributing sources have to be combined into a single presence information document. In addition, a script can modify, add, or remove information, force or block sending of status notifications. Operations may be customized with respect to the identity of receiving peers, i.e. a script may generate subscriber-specific views on the current presence status.

Several mechanisms for uploading scripts onto aggregation servers could be provided, ranging from HTTP upload using the PUT method to specific document manipulation protocols like WebDAV [3] or XCAP [13].

4. Script-based Presence Aggregation

Aggregation of distributed presence information can be done in several ways, depending on the desired level of abstraction. The choice may also influence the level of security that can be achieved. Automatic learning algorithms for which examples have been shown in section 2 deliver good results for detecting a suitable context description from a set of pre-defined values, e.g. to switch presentation of incoming messages from a handheld device to a workstation computer when entering the office.

Most of these application scenarios use a static description of available communication services, typically customized by the user who has entered a set of preference rules. Presence information is generated automatically from these preferences together with the known status of individual devices. Once this presence information is generated, a user has no option to customize this information prior to publication.

4.1. Controlling the Aggregation Process

The goal of our work is to provide users with a fine-grained control of the aggregation process. We have selected the JavaScript programming language for aggregation specifications to facilitate rapid adoption by users already having skills in development of applications for the World Wide Web. A custom runtime environment in the aggregation server provides objects and functions that are necessary to manipulate incoming presence status notifications and generate output to be published to subscribed watchers. The JavaScript-based *Presence Aggregation Language* (PAL) thus is used by an aggregation server to process incoming presence documents, to generate subscriber-specific presence documents and to specify when and where output documents should be published.

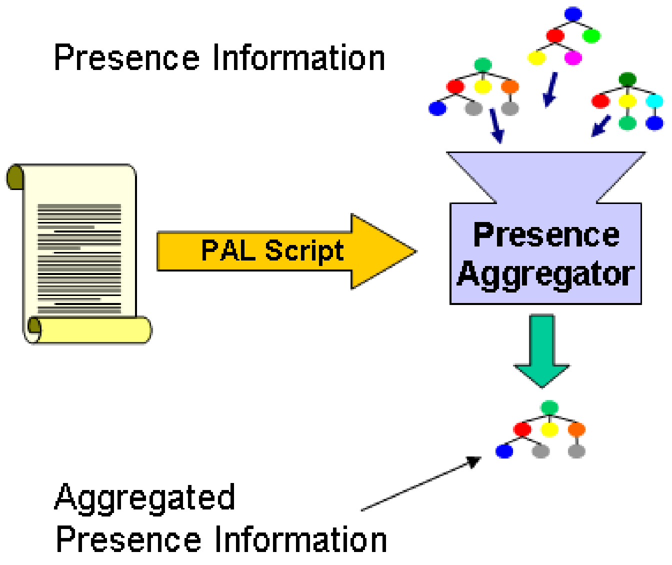

Figure 4: Application of PAL scripts on input documents.

Figure 4 shows the invocation of PAL scripts on incoming presence status notifications. The script operates on the logical structure of PIDF documents, here being visualized as trees. The script's output is a set of subscriber-specific documents that are sent to subscribed watchers according to the rules of the used presence transport protocol. PAL-specific filters can be installed at the input side to collect incoming presence documents until a notification matching a pre-defined pattern arrives or a script-specific timer expires. Additional mechanisms to control processing of presence notifications as event filters and throttling can be applied as well.

We use SIP-based event notifications [11], [14] for managing subscriptions as well as delivering presence information documents to subscribed watchers. Presence sources may publish status updates using the PUBLISH method described in [9] or via file upload. When the presence server is co-located with a registrar server, it may optionally create automatic subscriptions to registered SIP user agents to facilitate communication with applications that do not support the PUBLISH method.

4.2. Extending PIDF

Presence information is represented as XML using the Presence Information Data Format (PIDF) as specified in [10]. A PIDF document represents a set of *presence tuples* each of which describes a specific aspect of the publishing user's status. In order to give users more control over the aggregation process, we have extended PIDF to carry not only policy information specific to the corresponding document but also detailed descriptions of time-dependent values of presence attributes.

In particular, any presence tuple contained in a PIDF document may be augmented by meta-data giving application-specific information about the characteristics of that tuple. The meta-data is encoded in XML elements contained in a special XML namespace for our proposed extension. A simple type system facilitates development of generic aggregation engines and enables future extensions of our framework. Currently, the basic types `string`, `number`, and `boolean` are available. Complex types can be built as combinations of basic types, e.g. `list(T)` creates a homogeneous list containing only items of type `T`.

The meta-data added to the status description of a presence tuple (i.e. enclosed within elements of type `status` which are contained in `tuple` elements) can give additional information about the service capabilities of that particular communication channel. Let aside proper namespace declarations, a status description in a PIDF document therefore could look like this:

```
<pidf:status>
  <pidf:basic>closed</impp:basic>
  <media type="list(string)">audio, video</media>
  <incall type="boolean" timestamp="20040211203010"
          decay="linear(5)">true</pidfxy:incall>
</pidf:status>
```

The channel described in this example is declared to be closed by the element `basic` from the PIDF namespace. The following elements `media` and `incall` give additional information on that status description: First, the channel described here is capable of audio and video communication. Second, the device in this example is known to be in an active call since some specific point of time as indicated by the element `incall`.

Attached to a meta-data element, the `decay` can be used to describe the exactness of that information decreasing over time. The attribute's value denotes a mathematical function with optional parameters for better results. The value given here, `linear(5)`, refers to a mathematical function $linear_5(t)$ modeling a linear decrease of the exactness curve over time. $linear_k(t)$ then is defined in the interval $[timestamp, timestamp + \frac{1}{k}]$ as

$$linear_k(t) = 1 - \frac{(t - timestamp)}{k}$$

and $linear_k(t) = 0$ for any other t. As most processes are better approximated by an exponential function, we define another function $log_k(t)$ as

$$log_k(t) = e^{(-k * (t - timestamp))}$$

Finally, the function `const` can be used to specify a constant exactness value which is used primarily for backwards

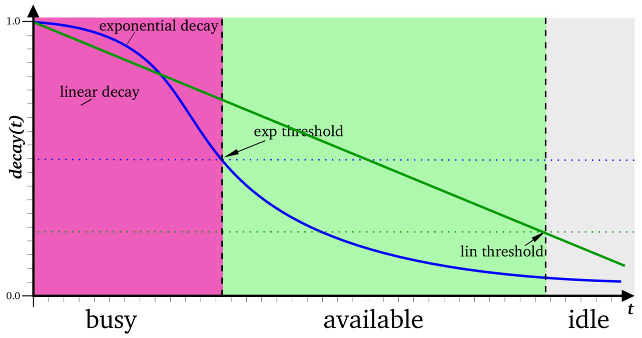

Figure 5: Applying decay functions to sensor values.

compliance with PIDF. Hence, `const` is the default for elements having no `decay` attribute.

Given this information, the sender's presence status can be calculated more accurately, especially in the case of lost update notifications due to network overload or event throttling being in effect. To do so, a threshold value must be set for each type of attribute that has a non-constant decay function. Once the value $decay(t)$ drops below that threshold for a time t, the presence status will be re-calculated, taking into consideration the changed exactness value (e.g. assuming the status `incall` has switched from `true` to `false`). Figure 5 illustrates this effect: On the y-axis, the exactness values for a linear and an exponential decay function are shown over time. Two threshold values are indicated by dotted horizontal lines, one associated with the exponential curve, the other with the descending line.

Suppose the PIDF processor has determined the presence status `busy` right after reception of the presence information. After some time—at the first vertical line—the exponential curve goes beyond its associated threshold value, and the presence status is re-calculated. As the corresponding status information (e.g. the `incall` attribute shown previously) is not considered up to date any more, the operation's result changes from `busy` to `available`, following the idea that the user has just put the phone on hook and is still in the office. As more time passes, this assumption gets less precise and might not be accurate at all. Therefore, another function is being used to trigger a re-calculation event at a later point in time (indicated by the second vertical line). Now, the processor cannot determine any activity, so the status is set to default (`idle` in this case).

This example clearly shows that appropriate threshold values as well as the mapping from input values to a valid presence status cannot be inferred automatically or determined from log analysis. Instead, the user has to specify a number of parameters that reflect personal preferences and device capabilities. The highly customizable aggregation system not only allows the specification of appropriate threshold values to trigger re-calculation of a presence status, it also lets the user specify her own function to determine the current presence status.

4.3. Implementation

In our implementation, the combination of meta-data attributes, decay functions and threshold values constitute a *trigger object* that is bound to a user-specific callback function. Once instantiated, the trigger is being watched by the runtime system of the aggregation engine. Whenever a trigger fires—i.e. when the associated decay value goes below the threshold value for the first time—its callback function is evaluated in a trigger-specific environment. The function is defined to take the current presence status, the updated decay values, and global system state as its input and returns zero or more presence descriptions. Any presence description that is returned will be published as extended presence information document to authorized subscribers according to the local policy of the server. Watcher-specific annotations of the generated presence information facilitate the generation of separate *views* on this document. A notification being sent to a particular watcher hence contains only data from the corresponding view. Any other information from the original status update notification is not made available.

We use an open-source JavaScript interpreter[1] with a customized runtime-environment to provide an object-oriented representation of PIDF documents as well as specialized functions to create, modify and remove trigger objects. The *Presence Aggregation Language* (PAL) thus consists of the JavaScript core and an additional function library. Every user is allowed to add PAL scripts to handle aggregation of any presence data related to her official presence URI (in many SIP implementations the corresponding address-of-record will be used). If no filters are in effect, the script will be evaluated for any (authorized) incoming presence notification related to this presence URI. The script output is a published as presence description to any subscribed watcher as mentioned before.

With PAL being Turing-complete, it is not possible to determine a script's complexity class or if script execution will finish at all. To achieve a considerable performance, the aggregation server must set an upper limit of CPU time to be spent by a PAL script. As with CGI scripts on HTTP servers, the presence server simply aborts processing after a pre-defined amount of time. A log entry is written to give the script author feedback of the problem. Unlike HTTP, clients cannot be notified of this event as there is no useful presence status to be published. However, the decay function in this case helps for subscribed clients not to rely on heavily outdated information.

[1]*SpiderMonkey* from the Mozilla project, see <http://www.mozilla.org/js/>.

5. Experience and Future Steps

Initial tests with our implementation of script-based aggregation have shown good results for small user bases. As expected, resource consumption (memory and CPU time) of the JavaScript interpreter makes this approach not adequate for high-performance installations. However, since strong trust relationships are required anyway, the aggregation service can be used in small installations, e.g. in a personal presence server that has to control the data of a limited number of principals. Script evaluation usually is bound to incoming presence update notifications, so the frequency of published status changes is a good performance indicator: As the corresponding SIP transactions must not overlap, we anticipate change notifications per user to occur less frequent than every 100 seconds. Given a script evaluation time below one second, this results to 100 users to be served in parallel. Additional load sharing mechanisms can be used to improve performance, so at least enterprise scale with thousands of users should be achievable.

A disadvantage of the script-based approach compared to automatic learning and rule-based actions is the lack of robustness. While predicting the results of the aggregation process is difficult for any of these techniques, scripts are the most brittle as small changes already may break the result. To mitigate this problem, we plan to provide a simulation framework that enables script testing using a large set of automatically generated test cases. To evaluate the results, the quality of the output documents must be determined. Factors that make a "good" output function yet have to be identified. In a future step, an adaptive environment could be added as well to provide automatic learning of script functions.

The modification of presence documents at intermediaries also requires the user to have a strong trust relationship with the aggregation server. Even worse, the server needs to sign notifications on behalf of the user to provide end-to-end security. For S/MIME, this would require the user to sign every outgoing notification with her private key, or the receiving entity must be configured to accept the server's signature instead of the user's. Moreover, to encrypt messages the recipient's public key must be available to the server. To facilitate private webs of trust, users must be able to upload the public keys of trusted subscribers to their dedicated aggregation server.

Besides the issues regarding secure communication, the aggregation server must be protected from scripts that prohibit proper operation of the service—either accidentally or on purpose. To avoid this, scripts should be executed in a sandbox that prevents any dangerous operation. Similar to common practice on WWW servers, scripts should also be killed after having consumed a certain amount of CPU time.

Finally, an interesting problem is handling presence update notifications that contain a delta encoding of what is considered the current presence status. As the aggregation service is defined on the complete status, the server has to re-construct this information prior to script evaluation. Unfortunately, if using multiple presence sources, the current status of every source has to be stored by the server to handle partial notifications from that particular source. The amount of memory to be allocated for a user hence grows linear to the number of contributing presence sources. Partial notifications therefore should not be used when aggregation is enabled.

6. Conclusion

We have presented our project for aggregation of presence information collected from multiple sources, e.g. mobile devices at different locations that have wireless connectivity. As an alternative to sensor fusion and automatic learning algorithms, our system makes use of user-specific scripts written in an imperative programming language. Presence attributes have been annotated with decay functions to model decreasing exactness of published information over time.

The language we have designed provides a persistent runtime environment that facilitates the presence status calculation using temporary values from previous execution cycles. Moreover, we have introduced the concept of *triggers* causing a callback function to be evaluated under certain conditions, with an updated presence information document as its result.

A major goal of our approach is to give users full control over the aggregation process and subsequent publication of resulting status documents. We anticipate that only highly skilled user will make use of this facility as software development is a difficult task, especially for distributed systems. Future improvements might include self-learning script functions and an integrated development environment that allows users with average skills to create and evaluate PAL scripts and adapt these with minimal effort as guided by the system. The integration of our script-based approach with self-learning systems facilitates development of presence aggregation services that combine the strengths of both worlds without giving up control.

References

[1] A. Carzaniga, “Architectures for an Event Notification Service Scalable to Wide-area Networks,” *PhD Thesis*, Politecnico Di Milano, 1998.

[2] J. Fogarty, J. Lai, J. Christensen, “Presence versus availability: the design and evaluation of a context-aware communication client,” *International Journal of Human-Computer Studies* , Vol. 61, No. 3, pp. 299-317, 2004.

[3] Y. Goland, E. Whitehead, A. Faizi et al., “HTTP Extensions for Distributed Authoring – WEBDAV,” *RFC 2518*, 1999.

[4] R. E. Grinter, L. Palen, “Instant Messaging in Teen Life,” *Proceedings of the ACM 2002 Conference on Computer Supported Cooperative Work*, pp. 21-30, 2002.

[5] Guanling Chen, D. Kotz, “Context Aggregation and Dissemination in Ubiquituos Computing Systems,” *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2002.

[6] T. Kanter, “Attaching Context-Aware Services to Moving Locations,” *Internet Computing*, Vol. 7, No. 2, pp. 43-51, 2003.

[7] M. Lonnfors, K. Kiss, “User Agent Capability Extension to Presence Information Data Format (PIDF),” *draft-ietf-simple-prescaps-ext-03*, Work in progress, 2005.

[8] J. C. McCarthy, A. F. Monk, “Channels, Conversation, Cooperation And Relevance: All You Wanted To Know About Communication But Were Afraid To Ask,” *Collaborative Computing*, Vol. 1, No. 1, pp. 35-60, 1994.

[9] A. Niemi (ed.), “Session Initiation Protocol (SIP) Extension for Event State Publication,” *RFC 3903*, 2004.

[10] J. Peterson: “Common Profile for Presence (CPP).” *RFC 3859*, 2004.

[11] A. B. Roach, “Session Initiation Protocol (SIP)-Specific Event Notification,” *RFC 3265*, 2002.

[12] J. Rosenberg, “A Data Model for Presence,” *draft-ietf-simple-presence-data-model-02*, Work in progress, 2005.

[13] J. Rosenberg, “The Extensible Markup Language (XML) Configuration Access Protocol (XCAP),” *draft-ietf-simple-xcap-06*, Work in progress, 2005.

[14] J. Rosenberg, “A Presence Event Package for the Session Initiation Protocol (SIP),” *RFC 3856*, 2004.

[15] J. Rosenberg, H. Schulzrinne, G. Camarillo et al., “SIP: Session Initiation Protocol,” *RFC 3261*, 2002.

[16] D. S. Rosenblum, A. L. Wolf, “A Design Framework for Internet-Scale Event Observation and Notification,” *Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 344-360, 1997.

[17] A. Schmidt, K. A. Aidoo, A. Takaluoma, et al. “Advanced Interaction in Context,” *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing* , pp. 89-101, 1999.

[18] H. Sugano, S. Fujimoto, G. Klyne et al., “Presence Information Data Format (PIDF),” *RFC 3863*, 2004.

[19] D. Zimmerman, “The Finger User Information Protocol,” *RFC 1288*, 1991.