A Client-Side Architecture for Supporting Pervasive Enterprise Communications

Amogh Kavimandan[†]
Department of EECS
Vanderbilt University
Nashville, TN 37235
a.kavimandan@vanderbilt.edu

Reinhard Klemm
Avaya Labs Research
233 Mount Airy Road
Basking Ridge, NJ 07920
klemm@avaya.com

Ajita John
Avaya Labs Research
307 Middletown Lincroft Road
Lincroft, NJ 07738
ajita@avaya.com

Dorée Seligmann
Avaya Labs Research
666 5th Ave.
New York, NY 10103
doree@avaya.com

Aniruddha Gokhale
Department of EECS
Vanderbilt University
Nashville, TN 37235
a.gokhale@vanderbilt.edu

Abstract

Pervasive, context-aware enterprise communications applications rely on detailed knowledge of people's contexts to facilitate efficient communications between people. We describe the challenges and intricacies in collecting and disseminating relevant user context data for the purpose of complex decision-making in enterprise communications applications. We show that the continuous and user-transparent collection of context elements such as user presence and activity enables certain simple decisions in communications applications while other decisions may depend on additional, communication task-specific user context details. These details can be acquired by applications as needed through what we call dialogs *in which users explicitly interact with communications applications. The rendering and processing of such dialogs is predicated on our notions of* device context *and* dialog presence. *We present the architecture and implementation of our client-side system* Argus *that addresses the context gathering and propagation challenges. We show how Argus, in conjunction with pervasive, context-aware enterprise communications applications, can accelerate and improve decision-making in an enterprise with a high level of convenience for users when establishing communications in response to enterprise events.*

Keywords

Availability, Context-Awareness, Dialog, Interruptibility, Pervasive Communications, Presence, SMIL, User Context, Web Browser, XUL.

1. Introduction

1.1 Motivation

Enterprise workflows automate various aspects of business processes and may be interspersed with communications with and between users. Examples of such communications include a conference call to resolve an open issue that was detected by an enterprise workflow or prompting a user for a decision on a given issue and collecting the user response through an HTML form. This type of communications can be modeled as communications applications on enterprise communications middleware. Enterprise communications applications establish communications sessions through various communication media and endpoints where enterprise users may collaborate with each other or with the sessions to provide information that serves as the basis for decision-making in the workflows. In the quest for productivity increases in enterprises it becomes important to establish communications with users virtually anywhere and anytime. Hence, pervasiveness of enterprise communications is a critical goal of communications applications.

Consider a scenario where a group of managers in a large and geographically distributed enterprise is on a conference call that deals with the breakdown of a vital assembly line in one of the enterprise's factories. Since this is a technically complex topic, the managers recognize the need for expert advice on assembly lines before making a decision. Suppose also that the enterprise has deployed middleware for executing pervasive, context-aware communications applications (*context-aware middleware*[1]) and that one of these applications is an expert finder application. An example of such a middleware is the *Hermes* [15, 16] system developed at Avaya Labs Research. A participant of the manager conference call would invoke the expert finder application through a Hermes user portal where he or she supplies the desired criteria for finding an expert, such as the required expertise, language, preferred

[1] In the remainder of this article we assume that context-aware middleware executes context-aware communications applications although generally such middleware support is not required.

[†] This work was done while the author was an intern at Avaya Labs Research.

communication modality (e.g. voice or instant messaging), desired time window for finding an expert, and a brief description of the discussion topic. A naive expert finder application could simply invite all matching enterprise associates that it finds in an enterprise directory to join the discussion, request a confirmation from each one to ascertain their availability for consulting with the managers, and ultimately select the first associate who confirms that he or she can participate in the manager discussion. However, there are several problems with this simplistic approach. The following list outlines some of these problems. In this list, we assume that Alice is a matching assembly line expert who is working from her home office and browsing the corporate newsletter through her Web browser at the time of the scenario.

1. *Invitation modality*: If the expert finder application picks a statically defined way of communicating the invitation to an associate – based on preferences for the associate or some hardwired decision – there is a good chance that the invitation does not reach its recipient within the desired time window because the recipient might be in a different location. For example, the expert finder application should not relay the invitation to Alice on her office phone because she is working from home. In fact, to ensure timely receipt and maximum convenience for the recipient, the expert finder application should relay the invitation to a communication endpoint that the recipient is known to be using at this point in time.
2. *Interruptibility*: If an associate is currently engaged in an activity that is more important to the enterprise than the participation in the manager conference, the expert finder application should not interrupt the associate. If Alice were currently collaborating with the corporate sales president on an important new sales strategy rather than browsing the corporate newsletter, the expert finder application might choose to skip Alice as a potential participant in the manager conference.
3. *Communication volume*: If the expert finder application generally invites all matching associates, regardless of current user contexts, the number of invitations that the associates would receive could prove a significant burden on productivity of the associates. Moreover, a large number of invitations and subsequent confirmations could lead to scalability problems for the enterprise communications platform.
4. *Confirmation modality*: If, after receipt of an invitation, an associate is forced to change communication modalities to provide the requested confirmation to the expert finder application, delays and inconvenience for the associate are introduced. For example, conveying the invitation to Alice as a voice message over her home phone but requesting the confirmation through her user portal would delay Alice's response to the confirmation request and cause inconvenience. In the worst case, the associate might not have access to the communication modality that is required to deliver the confirmation.
5. *Confirmation Negotiation*: In general, a confirmation request cannot simply be a yes/no prompt because the user needs the flexibility to negotiate details of the possibly ensuing interaction. For example, if Alice were willing to participate in the manager conference but needed to delay her participation by 20 minutes, the expert finder application should allow negotiating a delayed participation in the manager conference.
6. *Confirmation Adaptation:* The specific type of negotiation between the associate and the application needs to be technically supported by the associate's communication endpoint and device. For example, if Alice's browser were deployed on a cell phone with low-bandwidth data connectivity, the type of negotiation needs to be adapted to this specific device context.
7. *Negotiation Status:* The expert finder application needs to be able to track the progress that an associate makes with the negotiation. If, for example, Alice has forgotten to start the negotiation, the expert finder application might want to re-invite her to the conference call and send a new confirmation request, possibly through a different endpoint.

The problems raised by the naïve expert finder application can be avoided by inviting only matching associates who are currently present, available, and interruptible, and by subsequently engaging an invited associate in a negotiation dialog. We consider a user to be *present* if the user is logged into a session on a communication endpoint. *Availability* means the user has engaged in some activity within that session during a recent time interval. A user is *interruptible* if the recent or current type of user activity in that session allows an interruption by a communication request such as a request for information feedback. The negotiation dialog sent to an invited associate allows the latter to specify the actual availability for the manager conference. The dialog has to be composed in such a way that it can be rendered on the recipient's communication endpoint and device. Determining presence, availability, and interruptibility presupposes the collection of relevant user context data and its propagation to the pervasive and context-aware enterprise application. Note that negotiation dialogs provide a means for users to specify *application-dependent* user context parameters such as an appropriate time and communication modality for the given communication task, e.g. "I will be available for the manager conference in 20 minutes on my cell phone". In conclusion, the appropriate definition, collection, and propagation of user context data are of paramount importance to the execution of a pervasive, context-aware enterprise communications application such as the expert finder application.

1.2 Our Contributions

This paper motivates and details the challenges in gathering user context at communication endpoints for the purpose of supporting decision-making in pervasive and context-aware enterprise communications applications. It presents a client-side architecture and a prototypical implementation that enables a two-phase negotiation between an application and a user leading up to an interaction: (1) An implicit negotiation phase which relies on the user-transparent determination of the communication endpoints the user is using (endpoint presence) as well as the availability and interruptibility of the user at these endpoints. (2) An explicit negotiation phase where the user can specify and control an application-dependent part of the user's context through a multimodal dialog with the application. The specified part of the context allows the application to decide how and when the actual interaction takes place. We describe how our architecture collects *device context* to allow the pervasive, context-aware enterprise application to tailor dialogs to the recipient's specific endpoint and device capabilities and dynamic characteristics. We introduce the concept of *dialog presence* to allow such an application to track the current status of a dialog. Dialog presence can be used to, among many other things, trigger a reminder to the user to interact with the dialog if necessary.

We present our architecture in the context of user activity in a Web browser and explicit negotiations with the user using a multimodal dialog mechanism. The work highlights the specific pre-steps that need to be taken before drawing users into an interaction to guarantee efficiency and speed in pervasive, context-aware enterprise communications while maintaining the privacy and productivity of the users. We claim that these steps enhance the promise of pervasive services. Note that in this paper we do not focus on how user context information is leveraged in context-aware applications and middleware. For details on the usage of context in communications decisions in the Avaya Labs Hermes middleware we refer the reader to [15] and [16].

The remainder of the paper is organized as follows. Section 2 presents related work. Section 3 refines the challenges introduced in Section 1.1. Section 4 outlines the Argus architecture and shows how it resolves these challenges, especially for Web browsers as communication endpoints. We also detail some of the implementation work. Section 5 provides concluding remarks.

2. Related Work

User context includes any information that can characterize and convey a user's situation [1, 22]. The Active Badge System [26] is one of the earliest context-aware applications. It provides a location service to find people in an office environment, with the use of infrared signaling. Much research has been devoted to context-aware computation, communications applications and enabling platforms [6, 4, 7, 9, 27, 24]. An example of middleware specifically for context-aware communication applications is the Hermes platform developed at Avaya Labs Research and presented in [16]. Lei et al. discuss the notion of context-aware computation and describe a middleware for context gathering and dissemination in [19]. In the parlance of this reference, Argus can be seen as a context source that feeds context information gathered from Web browsers into a middleware for context-aware communications applications. In Argus we assume the presence of such a middleware and focus on a client that supports user context collection and dialogs.

MyVine [12] investigates the relationship between user presence on communication clients and user availability. The technology in [12] uses indirect ways of ascertaining a user's general availability such as speech detection, computer activity, calendar entries, and more. To determine a user's availability for a *specific* communication task, Argus first monitors user activity to measure the user's interruptibility for a dialog, then sends the dialog to the user, and finally evaluates the user feedback to the dialog. Notice that indirect availability sensing technologies can complement the Argus approach to user availability determination. An approach to determine a user's general presence and availability based on extrapolation of historical data is presented in [14].

The Cooltown project at Hewlett-Packard Laboratories introduced the concept of *Web presence*, which extends the "homepage" concept and bridges the World Wide Web and the physical world we inhabit [17]. It aims to make physical entities Web-present by embedding Web servers in them. In other words, physical entities gain a presence on the Web. This is essentially different from *Web browser presence* and activity introduced in our work, which determines whether a user is present in an active Web browser session and what specific activity the user is engaged in.

Begole et al. [2, 3] studied detailed computer activity coupled with information about user location, online calendar appointments, and e-mail activity. Begole et al. use temporal patterns in a user's presence to estimate when that user might be likely to return. They point out that a user who is usually present at a given time of day might be available at that time even if the user's computer has been inactive for several minutes. Argus collects detailed current user and device context, partially in a user-transparent manner and partially as a result of an explicit dialog with the user for a given communication task at this time. Argus propagates the collected data in a timely fashion to context-aware communications applications and delegates the context evaluation to these applications. Hence, Argus supports the establishment of communications based on *actual current* user context while not precluding the computation and use of temporal user context patterns described in [2, 3].

Collective Web awareness and collaborative browsing have been researched previously and some commercial products

exist in this area [25, 8, 29, 20, 10]. CoBrow [25] is a distributed tool that facilitates such collaborative browsing in the WWW. In CoBrow users with similar interests participate in virtual spaces built on the fly, so that they can learn and benefit together from individually gained knowledge of the shared interest. Similar to CoBrow and other efforts in collaborative surfing, Argus uses information such as what Web page a user is currently browsing. However, unlike CoBrow, the information is used for determining interruptability of the user. CoBrow aims to find users with common interests and facilitates collaboration between them. Argus on the other hand collects and propagates detailed user presence and activity information to the context-aware middleware. Moreover, in Argus the status information of a user is known only to the context-aware middleware, while this information is visible to all the other users sharing the same virtual space in CoBrow.

The Presence-Enabled Mobile Service (PEMS) in [23] is an example of a platform that integrates mobile devices such as cell phones with context-aware enterprise middleware. PEMS provides Mobile Presence Agents (MPAs) that can be deployed on mobile devices to detect the *device presence*, which is different from and complementary to Web browser presence.

3. Client-Side Challenges in Supporting Pervasive, Context-Aware Enterprise Communications Applications

In general, users frequently change their contexts over time. Among other things, users engage in varying activities, assume different communication modalities, and switch locations. Efficient and seamless execution of pervasive, context-aware enterprise communications applications therefore requires precise and timely determination of user contexts and in particular user presence and activities. Mechanisms deployed on user machines and devices are necessary to track and determine contextual information about users. Our scenario in Section 1.1 introduced several previously unaddressed challenges to client-side architectures for supporting pervasive, context-aware enterprise communications applications. In this section, we discuss these challenges in detail and outline the benefits of solving these challenges.

3.1 Collecting and Disseminating User Presence and Activity

Background: In the expert finder scenario, the notion of communication endpoint presence and activities is crucial for the effectiveness of the application. The application needs to be able to ascertain such context information about Alice to determine her availability and interruptibility for a conference invitation. The collection of detailed user context data at a communication endpoint far exceeds simple activity monitoring, say, in or on an operating system. The user context data that we want to define contains endpoint-specific usage information that allows inferring user presence, availability, and interruptibility. For example, the expert finder application needs to know whether Alice in her current Web browser session is reading her horoscope or whether she is logged into a Web-based conference bridge with other coworkers. In the former case, she is interruptible for the manager conference but perhaps not in the latter case. Furthermore, if the expert finder application ascertains that a user is present and available on a specific communication endpoint, it not only knows the user is present but also *how* to convey an important piece of communication to the user in a timely and convenient fashion. The client-side mechanism that we were envisioning must detect and collect detailed endpoint usage and forward it to the context-aware middleware in a timely fashion. This mechanism should be user-transparent and allow the user to protect his or her privacy by instructing the client-side mechanism to not propagate any or all presence and activity information to the middleware. Note that our presence and activity dissemination model is not peer-to-peer but follows a client-server approach where the middleware assumes responsibility for guarding or further relaying any presence and activity information from the client-side mechanism.

Problems: A number of questions arise in this context. For example, how do we define endpoint presence and activities and how detailed should such context information be? How can a client-side system gather context in a way that is transparent to the user? Where does context aggregation and evaluation take place? How often should context data collection take place? Which model, push or pull, should be used for data transmission? Some of these issues affect the timeliness and detail and thus the accuracy of the information that context-aware applications evaluate. Other architectural choices affect the scalability of the solution, the requirements for the bandwidth between the client and the middleware, and the computing load on the client.

3.2 Collecting and Disseminating Device Context

Background: In our scenario in Section 1.1, even if Alice's browser were equipped to handle a confirmation or dialog sent by the expert finder application there is no guarantee that her computing environment is suitable for receiving and rendering the dialog, allowing Alice to provide feedback, or return her feedback to the application. A pervasive, context-aware enterprise application has to be able to determine the capabilities of the platform or device that runs the user's endpoint in order to tailor the dialog to the device. For example, a dialog to be rendered on a cell phone-based browser might contain only simple graphics and allow the user to provide feedback by simply clicking on one of several form buttons in the dialog. Gathering and reporting client-side capabilities and parameters can also improve the quality of communication modality selection in the middleware. For example, if the middleware knows that Alice is currently using

a cell phone and needs to set up a conference with Alice as one participant it might decide against initiating, say, an instant messaging conference in favor of an audio conference. The instant messaging conference would require Alice to start an IM client on her cell phone and to type more or less complex phrases on her keypad. Similarly, dynamic parameters such as the current battery charge level on a portable device could be part of the data that a client reports to the context-aware middleware. In a variation on the previous example, if the application determines that Alice is present on a laptop but that this machine has only a few minutes of battery life left, the application might decide against an instant messaging conference in favor of an audio conference.

Problems: We need to define a user's *device context* that includes static and dynamic client-side capabilities and parameters that may be relevant to pervasive, context-aware enterprise communications applications. Dynamic device context parameters such as battery level or current data connectivity need to be collected and propagated to the context-aware middleware in a timely fashion. Our goal is to allow the application to infer the most appropriate communication modality and type of dialogs to be used for a certain user in a given situation. There are other problems that are analogous to those discussed at the end of Section 3.1.

3.3 Explicit Negotiation of User Context Parameters through Dialogs

Background: In our scenario, the ability of endpoints to receive, render, and return dialogs is as important as user presence and activity determination. In our expert finder example, just knowing that Alice is present, available, and interruptible does not imply she can or wants to participate specifically in the manager conference. What it does indicate is simply that Alice's Web browser may provide the most expedient means to deliver a piece of communication to Alice. A subsequent dialog would allow Alice to specify the part of her user context that is specific to the expert finder application and the request for Alice's participation in the manager conference. In particular, a dialog would allow her to negotiate her availability at a certain time with the expert finder application. The types of dialogs that we envision range from dialogs that purely display information to genuinely interactive and multimodal dialogs that can gather user feedback which the application can consider in subsequent decisions.

In our scenario, if Alice does not respond quickly to the dialog that the expert finder application sent to her, the application has no way of determining whether Alice received the dialog. For example, Alice may not be able to respond to the dialog immediately because she may have to check her calendar first and then ask her supervisor for approval to talk to the managers. In the absence of fast feedback from Alice, the application may choose to contact several other potential experts but such an approach is inherently inefficient because many potential experts might be excluded from consideration simply because they take time to respond to a dialog. Instead, we were looking for a way to collect and gather *dialog presence* that allows a context-aware application to determine whether a user has received a dialog. If necessary, the application could send a reminder to the user, through the same or an alternate endpoint, that asks the user to respond to the dialog.

Problems: A significant challenge involves identifying a dialog technology that is flexible enough to handle the requirements of a wide range of pervasive, context-aware enterprise communications applications. Although textual forms are simple and meet the basic requirement, we wanted to be able to include multimedia elements, for example video and audio clips. Such dialogs have to be asynchronously received and rendered in an off-the-shelf endpoint. The same endpoint needs to return user feedback to a dialog. Dialogs have to be designed by the developers of a context-aware application, so we are particularly interested in a standards-based dialog technology with a large tool base. The arrival of new dialogs in an active endpoint session has to be announced to the user. If a number of dialogs are sent to a user simultaneously or in an overlapping way they should be rendered in a meaningful, serialized fashion. Serialization of dialogs suggests categorization of dialogs based on priority or importance, where a higher-priority dialog gets precedence and is rendered before a lower-priority one.

The dialog presence concept raises its own problems. How do we best define and monitor dialog presence for the benefit of a

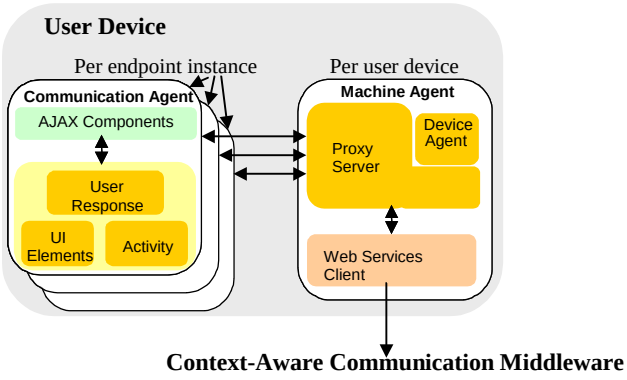

Figure 1: Argus System Architecture

wide range of context-aware communications applications? Unlike communication endpoint presence and activities or client-side capabilities, dialog presence changes must be propagated to the context-aware middleware with minimal latency.

3.4 Technical and Societal Merits of Resolving these Challenges

The challenges in the previous subsections were motivated by our scenario in Section 1.1 but we envision many other scenarios where context-aware communications applications

may greatly benefit from a system that meets these challenges. In our work, we aim at closing the loop from user context gathering at a communication endpoint to rendering dialogs initiated by a context-aware communication application in response to user context evaluation. Closing this loop can substantially accelerate and facilitate collaborative communications in enterprises and helps make enterprise communication processes seamless, both from a temporal point of view and from a user experience point of view. This is especially important for large, globally distributed enterprises where associates have varied and specialized skills and expertise, and where communication and collaboration are essential to fully utilizing the available human assets. In such environments we usually find diverse communication modalities with a wide range of communication endpoints, from video conferencing to instant messaging to various types of voice communication technology (POTS, VoIP, wireless, etc.). The devices and machines that host communication endpoints tend to be equally diverse and there is a considerable amount of uncertainty about which user is currently present on which endpoint, at which location, and what activities users are engaged in. Context-aware communication applications have to adjust to this type of environment to accelerate decision-making in enterprises and enhance workers' productivity.

4. Architecture and Implementation Considerations for Argus

This section presents the architecture and implementation of Argus, our Web browser-based client software for supporting context-aware communications applications. We first outline the reasons behind our choice of Web browsers as communication endpoints for context gathering and dialog rendering. We then describe how the Argus architecture and implementation meets the challenges and goals outlined in Section 3. Our Argus implementation at Avaya Labs interacts specifically with the Hermes context-aware middleware. In this section, however, we simply assume that the context-aware middleware provides a Web Services interface that allows Argus to report presence and user activity data as well as device context and that can relay dialogs originating at context-aware communications applications.

4.1 Web Browsers as Communication Endpoints

In the Argus project we decided to first focus on Web browsers as endpoints for collecting presence and activity information and rendering dialogs. In the interest of supporting pervasive communication services, we ultimately want to extend Argus to a wide spectrum of communication endpoints in order to collect user context at any endpoint that a user might employ and to render dialogs on any such an endpoint. Our specific interest in Web browsers for gathering user context, specifically presence and activity information, and for rendering multimedia dialogs is motivated by our observation that Web browsers are universally deployed across user computers and devices, are very versatile and programmable software components with the ability to render multimedia content, and have turned into the client software of choice for many user activities, from access to collaboration tools to information retrieval to downloading multimedia content. Because many enterprises have deployed Web-based enterprise portals that provide a unified access point to a wide spectrum of applications in the enterprise [18], Web browsers have gained particular importance as client software in enterprises. Thus, many enterprise users spend a lot of time interacting with Web browsers, and context data collection through Web browsers becomes meaningful. Web browsers already support uploading of information to servers and the retrieval of multimedia content, thus providing the infrastructure for transmitting presence and other user context data to context-aware applications, retrieving dialogs, allowing user interaction with such dialogs, and returning user feedback to context-aware applications. Hence, browsers can enable a seamless user experience for accessing Web content and interacting with context-aware applications through dialogs.

4.2 Argus Architecture

Figure 1 shows the Argus architecture. Argus contains two major components, a *communication agent* and a *machine agent*. The communication agent is tightly integrated with a Web browser. It allows Argus to observe a user's Web browser activities and enables the injection of dialogs into the Web browser. The communication agent consists of browser User Interface (UI) components, AJAX (*A*synchronous *J*ava *a*nd *X*ML) components [13], an activity monitor, and a User Response module. More than one communication agent may be active on a given user device, depending on how many browser instances are currently executing, but there is only one machine agent per device. The machine agent resides outside the browser and mediates between the communication agents on a device and the context-aware middleware. The machine agent contains a *proxy server*, a *device agent*, and a Web Services client for communication with the context-aware middleware. In the following subsections, we will explain the rationale behind our particular architecture and show how the various components act in concert with a Web browser and the middleware to support context-aware communications applications.

4.3 Collecting and Disseminating User Context

4.3.1 Presence and Activity Definition

In Argus, a Web browser *session* starts when an instance of the browser begins executing and ends when that instance terminates. Note that a Web browser session corresponds to our notion of user *presence* introduced in Section 1.1. Within a session, each user interface activity, in particular browser button clicks, hyperlink clicks, changing browser window size, entering text into a form or clicking buttons in a form, and

rendering multimedia content, that can be gleaned from the browser counts as a user presence indication and is captured as user activity. User interface activities are indications of user *availability* as introduced in Section 1.1. Note that this approach is different from a publisher-subscriber system, wherein various subscribers register for *specific* events and receive notifications of such events generated by publishers. In Argus, the context-aware middleware is the *only* recipient of user context information and receives *all* such information. Context-aware applications categorize the types of presence and activity data reported by Argus and subsequently make a determination of the user's presence, availability, and interruptibility based on the detailed presence and activity information. Interruptibility of a user has to be assessed vis-à-vis a specific type of application and a specific piece of communication or interaction within the application.

4.3.2 Presence and Activity Collection

Our choice of a Web browser as a presence and activity collection endpoint and dialog rendering device had a strong impact on the entire Argus architecture and implementation. One of our criteria in choosing a Web browser was ease of implementing context data collection and dialog rendering. Moreover, rather than designing a special-purpose Web browser, we intended to use an off-the-shelf Web browser with a large install base and modify it for our purposes. We decided to use the Mozilla Firefox Web browser [30] customize it through its extension mechanism.

Extensions provide a modular and platform-independent way to add new functionality to Firefox. The design of an extension is event driven, where the user interface is specified using the XML User Interface Language (XUL) [31], while the event handlers that define the browser behavior are implemented using JavaScript. One of our goals in designing Argus was user convenience, in particular the user does not have to explicitly specify presence and activity information when he or she is using the browser interface. The *Activity* part of our Firefox extension shown in Figure 1 automatically monitors the user's activities in a browser window. Beginning with the start of each browser session, every user interface activity triggers an event, and a corresponding JavaScript handler inside our Firefox extension is called that logs the event. The event handler execution is not noticeable to the user. Periodically, with a user-configurable frequency, the Argus Firefox extension bundles the logged events into a presence and activity information packet and pushes it to the machine agent. Obviously, the higher the presence and activity update frequency is, the more accurate the presence and activity information is that context-aware applications may evaluate. On the other hand, a higher update frequency also increases the computing load on the client that hosts Argus and it may increase the bandwidth requirements between Argus and the middleware.

To allow the user to protect his or her privacy, the UI elements shown in Figure 1 provide a toolbar in every browser window that contains buttons to turn context gathering and dissemination on or off. If the user allows Argus to propagate context data to the middleware, it becomes the responsibility of the middleware to protect the user's privacy. For example, the Hermes system introduces the notion of *brokered presence* where context-aware applications make communications decisions based on user context but do not necessarily disclose user context to other users.

4.3.3 Context Propagation

For a given client, we assign the context propagation functionality to a dedicated machine agent, specifically an HTTP proxy server which acts as a broker for all sessions between that endpoint and the context-aware middleware.

The following are advantages of separating the machine agent from the presence and activity collection mechanism, i.e. the communication agent:

- *Separation of concerns.* The system design is modular since collection and propagation functions are kept separate. This also makes the extension simple in design.
- *Collective state management:* The machine agent provides state management for all active sessions on a single client. The state of an active session is a measure of how far a user dialog has proceeded in that session (see Section 4.5.2). A number of active sessions may be present for a single endpoint and a state has to be recorded for each of those active sessions. In the interest of scaling the Argus approach to many clients and communication endpoints, we decided to perform state management on the client side rather than in the context-aware middleware.
- *Device context*. As we will see in Section 4.4, deploying a machine agent outside the browser enables the gathering and propagation of device context.
- *Efficiency.* While it was necessary and appropriate to use a Firefox extension for user context gathering and dialog injection, the Firefox extension model is very restrictive and inefficient for general purpose computing tasks such as the propagation of user context to the middleware. Placing non-user interface functionality into a separate machine agent therefore simplifies the Argus design and enhances its efficiency.

Propagating presence and activity information from the communication agent to the middleware involves the following steps:

1. The communication agent converts JavaScript event handler function calls into asynchronous HTTP requests for the proxy server.
2. If the proxy server receives a Web browser startup event notification from the communication agent, it invokes a Web Services (WS) method in the middleware to obtain a session ID. The session ID will identify this particular browser instance as a context data source and allow Argus

to associate context data sources with presence and activity events when communicating with the middleware in the future. The session ID is also used when the machine agent polls the middleware for pending dialogs (see below).

3. The proxy server invokes a WS method in the middleware through the WS client interface and reports the latest user context update from the communication agent to the middleware.

AJAX is used for the interaction between the communication agent and the machine agent as shown in Figure 2. The asynchrony provided by AJAX ensures complete user transparency of the presence and activity collection mechanism. The interaction between the communication agent and machine agent on the one side and the middleware on the other side is two-way, i.e. while the JavaScript event handlers make asynchronous HTTP requests, the middleware sends SMIL dialogs (see below) to the communication agent through the proxy server.

A timeout event triggers the periodic presence and activity propagation mechanism in the Argus Firefox extension, illustrated by the JavaScript (JS) calls in Figure 2. The AJAX routines convert JS native calls into HTTP requests, to be sent to the machine agent. Once the AJAX routines have created the server request for the proxy server, control is returned back to the extension event loop to handle further events in the browser.

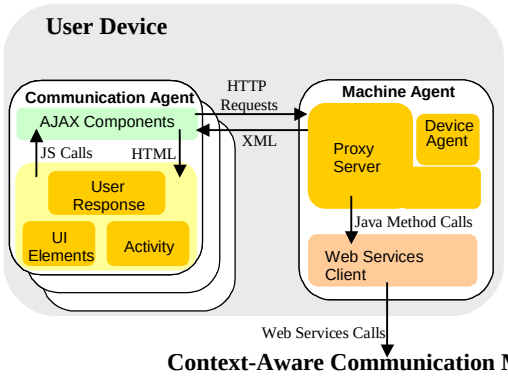

Figure 2: Component Interaction in Argus

4.4 Collecting and Disseminating Device Context

Because the machine agent resides outside a Web browser, we can use it to gather device-wide client platform/hardware capabilities and parameters and propagate that information to the middleware. Section 3.2 introduced the notion of device context as a collection of such capabilities and static/dynamic parameters. The device agent inside the machine agent implements this functionality. It is a simple operating system-specific module that reports static client-side capabilities to the middleware once upon startup of the agent. Throughout its execution, it periodically updates the middleware with dynamic client-side parameters. Currently, the device agent is implemented for Windows XP but it can be easily modified for other operating systems. The client-side capabilities and parameters sent to the middleware include the type of machine (laptop, desktop, PDA, mobile phone, etc.), processor information, network interface type, type of keyboard and mouse if any, battery strength remaining, data connectivity status, etc. Because the device agent is implemented as part of the machine agent rather than as part of the communication agent, it can work independently of any browser sessions and send client information directly to the middleware.

4.5 Explicit Negotiation of User Context Parameters through Dialogs

4.5.1 Dialog Technology

Argus uses the Synchronized Multimedia Integration Language (SMIL) [32], a W3C proposal, originally designed for describing and rendering interactive audiovisual presentations. Because SMIL supports interactions with users and rich multimedia content, in particular audio and video clips in addition to textual data and forms, it lends itself to our notion of dialogs. We also chose SMIL as our dialog technology of choice because (a) it is flexible in that it can be easily used for a wide number of context-aware applications, (b) a large variety of tools to create SMIL dialogs is available, and (c) SMIL can be used as a general way of defining structured dialogs irrespective of the endpoint modality of communication. Note that SMIL is an XML-based language, and hence Argus uses the same format for dialogs that it uses for data exchange (through Web Services) between Argus agents and the context-aware middleware.

4.5.2 Dialog Injection and Rendering

The following are the steps involved in injecting a dialog, originating at a context-aware communication application, into a Web browser session:

1. The context-aware application associates the dialog with the session ID of the target Web browser session.
2. The machine agent detects a nonempty list of dialogs pending in the middleware for the active session. To this end, the machine agent calls the appropriate WS method in the middleware and uses the ID of the active Web session as a parameter for the WS method.
3. The machine agent receives the list of pending dialogs for this user for the active Web browser session from the middleware.
4. The machine agent performs delivery of the dialogs to the communication agent, one dialog at a time.
5. The communication agent, through a Firefox extension shown as the User Response component in Figure 2, retrieves the dialog and Firefox presents it to the user through a pop-up window.

If more than one dialog is pending for this user and the given Web session, the dialog with the highest priority, as assigned by the originating context-aware application, is chosen by the machine agent for delivery to the communication agent. The machine agent sends the SMIL dialog to the communication agent of the respective Web session, which renders it as an HTML form, possibly with multimedia content. From the user's perspective, the arrival of a dialog is indicated by a pop-up browser window that displays the SMIL content. In some applications, a single interaction between a user and a context-aware communications application consists of a number of dialogs where the next dialog served may depend on the user response to an earlier dialog. The interaction can be seen as a graph, with dialogs as its nodes and edges as user responses to dialogs. In such cases, the state of a particular interaction graph is maintained at the machine agent, the state being last dialog sent. The interaction completes when no dialogs remain in the interaction graph.

4.5.3 Dialog Feedback

Since a dialog is rendered as an HTML form, any user response is collected by the Web browser and sent back to the machine agent as an HTTP request. The machine agent may use this data to determine which dialog in the current interaction to send next, or forward it to the middleware for further processing.

4.5.4 Dialog Presence

When a context-aware communications application prompts the Hermes middleware to deliver a dialog to a user, Hermes sets the presence status of the dialog to *pending*. When Argus polls the context-aware middleware for pending dialogs for this user and receives a list of one or more dialogs, the middleware knows that Argus has received these dialogs and assigns a presence status of *consuming* to these dialogs. At this point in time, Argus is rendering the dialog to the user. Once a user has responded to the dialog via Argus, Argus returns the user responses to the middleware and the middleware can again adjust the presence status of the dialog to *completed*.

In the future, we envision a class of dialogs that are not immediately rendered upon receipt by Argus and therefore do not interrupt the recipient. Instead, Argus would simply display a *dialog waiting* icon and only when the user actually starts interacting with dialogs would Argus notify the middleware of the beginning of the dialog consumption through the user. The timing of dialog presence status changes on the middleware side would be adjusted correspondingly, thus enabling a more accurate determination of when a user is actually interacting with a dialog.

5. Summary and Outlook

In this article, we motivated and discussed in detail the conceptual and design challenges for clients that are intended to support pervasive, context-aware enterprise communications applications. We introduced the notion of a two-phase negotiation between such clients and applications in order to determine the availability of a user for a specific communication task (such as a conference call) initiated by an application. The first phase is an implicit, user-transparent, and automatic gathering of user context data at user communication endpoints. It allows the application to determine the presence, availability, and interruptibility of the user for a feedback request. If a user is deemed present, available, and interruptible, the second phase occurs and engages the user in an explicit dialog that allows the user to provide feedback to the application with an indication of his or her actual availability for the communication task.

We described the architecture and implementation of our client-side system *Argus* that collects user context data, specifically user presence and activities, and propagates it to context-aware communications middleware. In its current form, Argus targets the Mozilla Firefox Web browser as a widely deployed and technically versatile communication endpoint for the collection of user context data (first negotiation phase). We defined multimodal SMIL-based dialogs (second negotiation phase) that can be rendered in a Firefox browser through mediation in Argus. Based on static and dynamic client-side parameters and capabilities (*device context*) that Argus collects and disseminates to the middleware, a communications application can tailor a dialog so that it can be rendered on the recipient's device. We demonstrated through a sample scenario that the combination of user context gathering at an endpoint such as a Web browser and the rendering of interactive SMIL dialogs can greatly aid pervasive enterprise communication applications in accelerating and facilitating decision-making in enterprises.

The next steps in the Argus project are extensive user studies in the system. We intend to investigate the scalability of our solution, measure the latency and accuracy of user context data as collected by Argus, and determine through a user study how effective Argus is in meeting the goals and challenges outlined in Sections 1 and 3. We already have extended Argus to work with the Thunderbird [28] mail client application and, in the future, also want to extend Argus to a wide spectrum of communication endpoints. Examples of endpoints that we are considering are softphones and IM clients.

6. References

[1] Abowd, G. D., Dey, A. K. Towards a Better Understanding of Context and Context-Awareness. *Proceedings of 1st International Symposium on Handheld and Ubiquitous Computing (HUC 99), Lecture Notes in Computer Science*, no. 1707, Springer-Verlag, Heidelberg, Germany, 1999, pp. 304-307.

[2] Begole, J. B., J. C. Tang, R. B. Smith and N. Yankelovich. Work Rhythms: Analyzing Visualizations of Awareness

Histories of Distributed Groups. *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 2002.

[3] Begole, J. B., Tang, J. C., and Hill, R. Rhythm Modeling, Visualizations, and Applications. *Proceedings of the ACM Symposium on User Interface Software and Technology*, 2003.

[4] Campbell, R. H., Ranganathan, A. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, June 2003.

[5] Chaari, T., Laforest, F., and Celentano, A. Service-Oriented Context-Aware Application Design. *Proceedings of the International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP)*, Ayia Napa, Cyprus, May 9, 2005.

[6] Chen, H. An Intelligent Broker Architecture for Context-Aware Systems. PhD proposal in Computer Science, University of Maryland Baltimore County, USA, January 2003.

[7] Chen, H., Finin, T., Anupam, J. An Ontology for Context-Aware Pervasive Computing Environments. *Workshop on Ontologies and Distributed Systems*, IJCAI-2003, Acapulco, Mexico, August 2003.

[8] Cohen, D., Jacovi, M., Maarek, Y. and Soroka, V. Livemaps for collection awareness. *International Journal of Human-Computer Studies 56, 1 (January 2002), 7-23.*

[9] Dey A. K. Providing Architecture Support for Building Context-Aware Applications. PhD thesis, November 2000, Georgia Institute of Technology.

[10] Day, M. and Foley, S. Selective dissemination of information in a colleague awareness application. *The Demonstration Session of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW)*, Seattle, WA, November 1998.

[11] Day, M., Rosenberg, J., Sugano, H. A Model for Presence and Instant Messaging. RFC 2778, February 2000.

[12] Fogarty, J., Lai, J., and Christensen, J. Presence Versus Availability: The Design and Evaluation of a Context-Aware Communication Client. *International Journal of Human-Computer Studies* 61, 3 (September 2004), 299-317.

[13] Garrett, J.J, Ajax: A New Approach to Web Applications. *Adaptive Path Essays*, www.adaptivepath.com/publications/essays/archives/000385.php, February 2005.

[14] Horvitz, E., Koch, P., Kadie, C.M., and Jacobs, A. Coordinate: Probabilistic Forecasting of Presence and Availability. *Proceedings of the Eighteenth Conference on Uncertainty and Artificial Intelligence*, Edmonton, Alberta, July 2002, 224-233.

[15] John, A., Klemm, R., Mani, A., and Seligmann, D. Hermes: A Platform for Context-Aware Enterprise Communication. *Third International Workshop on Context Modeling and Reasoning (CoMoRea)*, March 13-17, 2006, Pisa, Italy.

[16] John, A., Klemm, R., and Seligmann, D. Hermes: A System for Orchestration of Shared Communication Spaces. *Technical Report ALR-2004*-019, April 2004.

[17] Kindberg, T., et al. People, Places, Things: Web Presence for the Real World. *Proceedings of ACM Mobile Networks and Applications Journal*, 2002.

[18] Knorr, E. The new enterprise portal. *InfoWorld* January 2004, http://www.infoworld.com/article/04/01/09/02FEportal_1.html, on 10/31/2005.

[19] Lei, H., Sow, D., Davis, J., Banavar, G. and Ebling, M. The Design and Applications of a Context Service. *ACM SIGMOBILE Mobile Computing and Communications Review 4*, 6 (October 2002), 45-55.

[20] Moody, P. WebPath: synchronous collaborative browsing. *Demonstration Session of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW)*, Seattle, WA, November 1998.

[21] Padovitz, A., Loke, S., and Zaslavsky, A. Towards a Theory of Context Spaces. *Workshop on Context Modeling and Reasoning.* Orlando, Florida, March 2004.

[22] Ranganathan, A. and Lei. H. Context-Aware Communication. *IEEE Computer 4*, 36 (April 2003), 90-92

[23] Shan, X. A Presence-Enabled Mobile Service Platform for Integrating Mobile Devices with Enterprise Collaborative Environment. *International Workshop on Wireless Ad-Hoc Networks*, May 2005.

[24] Smailagic, A., Siewiorek, D., Anhalt, J., Gemperle, F., Salber., D. and Weber, S. Towards Context Aware Computing: Experiences and Lessons Learned. *IEEE Journal of Intelligent Systems 3*, 16 (June 2001), 38-46.

[25] Sidler, G., Scott, A. and Wolf, H. Collaborative browsing in the world wide web. *Proceedings of the 8th Joint European Networking Conference, Edinburgh, 1997.*

[26] Want, R., Hopper, A., Falco, V., and Gibbons J. The active badge location system. *ACM Transactions on Information Systems*, January 1992.

[27] Yau, S., Karim, Wang, Y., Wang, B., and Gupta, S. Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *IEEE Pervasive Computing, joint special issue with IEEE Personal Communications*, 1(3), July-September 2002, 33-40.

[28] http://www.mozilla.org/projects/thunderbird/ on 10/31/2005.

[29] http://www.lotus.com/sametime, on 10/31/2005.

[30] The Firefox Project. http://www.mozilla.org/projects/firefox/, on 10/31/2005.

[31] The XML User Interface Language (XUL) 1.0. http://www.mozilla.org/projects/xul/xul.html, on 10/31/2005.

[32] Synchronized Multimedia Integration Language (SMIL 2.1). http://www.w3.org/TR/SMIL2/, on 10/31/2005.