Enhancing Enterprise User Productivity with Embedded Context-Aware Voice Applications

Amogh Kavimandan[‡], Reinhard Klemm[†], Aniruddha Gokhale[‡], Dorée Seligmann[†]
[‡] *Department of EECS Vanderbilt University, Nashville, TN 37235*
[†] *Avaya Labs Research, 233 Mount Airy Road, Basking Ridge, NJ 07920*

Abstract

The increasing amount of software and computational capabilities in voice endpoints, switches, and networks creates an opportunity for embedding advanced applications in voice communication paths. Particularly in an enterprise environment, such applications can alter the traditional behavior of voice communications from simply connecting two or more people to software-assisted connection establishment and enhanced on-call features. In this paper, we claim that embedding context-aware applications in communication paths can greatly increase the efficiency, effectiveness, and convenience of enterprise communications and thus the productivity of enterprise users. We identify the challenges associated with embedding context-aware applications in communication paths and exemplify our central claim by presenting a context-aware voice communications application CallerID++. *CallerID++ aims at improving the aged concept of caller ID as the basis for a callee's decision to accept or reject an incoming call. Through a call acceptance negotiation between caller and callee prior to call establishment, CallerID++ allows the callee to assess the importance of an incoming call relative to the callee's current activity. CallerID++ thus helps minimizing unwanted interruptions without rejecting important calls.*

1. Introduction

Telecommunication between people is experiencing a fundamental transformation due to a variety of factors including the proliferation of communication devices and media, the increasing mobility and computational power of communication endpoints, and new network-level technologies such as Voice Over IP. However, in the realm of telephony, this technological revolution has so far not resulted in an equally revolutionary change in the *processes* that people employ to connect with each other. In other words, voice communication processes have not kept pace with the technological improvements in telephony endpoints and devices, networks, and switches. For example, while presence-enhanced softphone features are state-of-the-art in enterprise voice communications, it has not eliminated the phenomena of phone tag, unwanted interruptions of busy users through incoming calls, or busy users rejecting incoming calls that later turn out to be important [4]. Such incidents hamper the efficiency, effectiveness, and convenience of enterprise voice communication processes as we will show in two scenarios below. Due to their impact on user productivity and thus enterprise competitiveness, resolving these impedances in communication processes in general is extremely important. Improvements to *voice* communication processes gain particular importance because of the pervasive nature of mobile and stationary telephony in modern enterprises.

Scenario 1. Consider a scenario where a group of employees is on a conference call about an important technical issue in a new software release. During the call, one of the participants, Alice, receives a phone call from her supervisor Bob. Assuming that this is an important call, Alice interrupts her participation in the conference call and switches to the incoming call from Bob, only to realize that Bob wants to simply talk about her availability for their next monthly project meeting. Alice finishes her conversation with Bob as quickly as she can and rejoins the conference call. Note that Alice based her decision to take Bob's call on Bob's caller ID and by inferring the importance of the call from Bob's position as Alice's supervisor. However, considering that Bob only wanted to talk about a regularly scheduled event, whose importance is much lower than that of the ongoing conference call, Alice might have preferred to let Bob's call go to voice mail and deal with it later instead of temporarily leaving the conference call. We may safely consider the above scenario an *unwanted interruption* of work, which may affect the efficiency, efficacy, and convenience of the entire team on the conference call.

Scenario 2. Now consider a different scenario where Alice and Bob are in a lengthy discussion about the previous night's football game in Alice's office. At this time, Cynthia, a software developer working on a project managed by Alice, suspects that she may have discovered a new security flaw in the software. She calls Alice to find out whether Alice is aware of this flaw. Assuming that a call from Cynthia at this time is not overly important and because Alice does not want to appear rude to her supervisor, Alice chooses to let Cynthia's call go to voice mail and continues talking to Bob. After a prolonged chat, Bob leaves Alice's office. Alice attempts to call Cynthia back but by that time Cynthia has already left her office for her lunch break. Alice only manages to reconnect with Cynthia much later. Clearly, there are two problems with this scenario. One is that Alice decides not to take Cynthia's call under the erroneous assumption that the call is not of higher importance than her interaction with Bob. Secondly, Alice and Cynthia enter a phase of phone tag. Both Alice's and Cynthia's response time in this scenario, speed of decision-making, convenience, and productivity are negatively affected.

In general, although it is agreed that enterprise users may benefit from knowing the purpose of a call in addition to caller ID and cues about recipient's availability and interruptibility [8, 9], very little work has been done to incorporate a corresponding mechanism into the communications infrastructure. In this article, we claim that embedding context-aware applications in call paths can significantly increase the efficiency, effectiveness, and convenience of enterprise voice communication processes. We identify the challenges associated with, and requirements for embedding context-aware applications in call paths. We exemplify our claim in the voice communication domain by presenting *CallerID++*, an embedded context-aware application. Its goal is to improve enterprise user productivity by minimizing the number of unwanted user interruptions and the number of important calls that callees let go to voice mail because of misjudging the call importance. Moreover, CallerID++ helps reducing the frequency of phone tag.

The remainder of this paper is organized as follows. Section 2 elaborates on the challenges in realizing the goals of the CallerID++ application; Section 3 shows the behavior of CallerID++ and how its operation meets the requirements and challenges listed in Section 2; Section 4 outlines the CallerID++ architecture and implementation; Section 5 describes related work. We provide concluding remarks in Section 6.

2. Challenges and Requirements for the CallerID++ Application

Traditional caller ID can be considered a simple application that is embedded in the communication path between a caller and a callee. A callee can use caller ID to decide whether to accept an incoming call. As we saw in the two scenarios in Section 1, however, caller ID may be a poor indicator of the importance of an incoming call relative to the importance of the callee's current activity. In other words, caller ID provides little useful decision support for call acceptance to the callee. On the other hand, the caller has no indication of the callee's current availability for the intended call and therefore the most expedient way to find out whether the callee is available for a conversation with the caller at this point in time is to actually place the call. In other words, traditional telephony provides no decision support for call placement to the caller either. Even state of the art presence-enhanced telephony tools [4, 7] fare only marginally better than caller ID. For example, in the two scenarios in Section 1, Alice was *present* in both scenarios but *available* only for the incoming call in the second scenario. Some presence-enhanced telephony tools [6] include current call activity in the user's presence status, for example *on call*. While this may have prevented Alice's unwanted interruption in Scenario 1, a change in that scenario demonstrates the limits of this approach. Suppose Alice receives a call from Bob not about the next monthly project meeting but about an outage of a company product at a company's customer. We can assume that Alice would want to interrupt her participation in the conference call and take Bob's call to consult with him on the product outage because the outage may be of higher importance than the current conference call. Alice is on a call at this time and yet she is available for Bob in this modified scenario. As Wiberg and Whittaker [8] note, caller ID does not convey the purpose of a call and there is little direct support for availability management in telephony, resulting in unwanted interruptions and tardy responses to important calls that the callee lets go to voice mail.

Based on the above observations, we set out to design an application that improves upon the shortcomings of caller ID on the one hand and presence-enhanced telephony tools on the other hand. The following is the list of challenges and requirements that we intended CallerID++ to meet.

1. Reducing the number of unwanted interruptions and missed important calls. As our scenarios demonstrate, the drawback of caller ID is its inability to convey anything about the call itself. CallerID++ therefore has to be able to collect a *call context* from the caller, i.e. call parameters including perceived importance, topic, and desired response time window, and convey it to the callee. The call context should be detailed enough to provide the callee with a real sense of the call intent and urgency. In addition, the call parameters to be collected should themselves be configurable, so that context collection policies could be defined depending on specific enterprise and group needs, and chosen dynamically at the time of call establishment. However, as Item 3 below implies, gathering the call context should take as little time and effort as possible.

2. Phone tag reduction. If the callee does not accept an incoming call immediately, CallerID++ should help the callee with reconnecting with the caller at a later time if so desired. Note that “later” can be any time between a few seconds to a few days from now or more. This goal may require the determination of the caller’s and the callee’s *user contexts* at a later time. The context of a user contains a description of currently observed activities of the user including communication activities (phone calls, Instant Messaging (IM) chats, email activity, etc.) and transcends mere user presence. Evaluating the caller’s and callee’s contexts helps finding a time at which both users are likely to be available for re-establishing the call.

3. Call reception feedback. CallerID++ has to provide the callee with the option of signaling the receipt of a call attempt back to the caller. Otherwise, the caller has no indication of whether the callee actually became aware of the call attempt. Without such feedback from the callee, the caller may unnecessarily try to reach the callee on alternate endpoints. This phenomenon is somewhat similar to phone tag and equally undesirable.

4. Call encouragement. Many presence-enhanced tools graphically display the presence or availability status of a user to a communication initiator, for example a caller. Such tools put the burden of determining the recipient’s interruptibility for the initiator’s contact attempt on the initiator [3, 10, 11] and often discourage the initiator unnecessarily from contacting the recipient at this point in time. Moreover, disseminating a user’s detailed presence and availability status to other users, which would be necessary so that an initiator can properly judge the recipient’s availability, may constitute an undesirable disclosure of very sensitive data. In contrast, we strongly prefer for the callee to determine whether she is available for the caller for the given purpose of the communication attempt.

5. Effective call context rendering: When CallerID++ renders the call context on a callee endpoint the choice of an endpoint and call context presentation may be made dependent on the callee’s user context. If the callee is already on a phone call and is known to be present but not actively engaged on another endpoint that can render the call context, for example an IM client or a Web browser [16], it is preferable to route the call context to that endpoint. Otherwise, the callee’s ongoing phone conversation would be disturbed. If the evaluation of the callee’s user context does not turn up an alternative endpoint, the call context is best rendered through the callee’s phone. Depending on the capabilities of the callee’s phone and the current phone status of the callee (being on a call or not), the call context can either be rendered on a textual display on the phone or as an audio overlay (*whisper*) on an ongoing phone call.

6. Pervasive deployment. Just as traditional caller ID is part of the telephony fabric and thus available to users most of the time, without the need for explicit activation or execution, CallerID++ needs to be embedded in a voice communications system and activated simply through call placement. We are primarily interested in improving the user voice communication experience in an enterprise environment, but the same idea could be applied to a public telephony network.

7. User convenience. The primary goal of CallerID++ is to increase user productivity. Hence, CallerID++ must be easy to learn and use for both callers and callees. The extra effort that CallerID++ requires from either the caller or callee must be minimized.

One aspect of the call establishment process that we have not dealt with yet is the potential misuse of the CallerID++ capabilities. For example, it is easy for the caller to deliberately underreport the expected call duration or to assign the highest possible importance rating to the call during call context collection and thus trick the callee into accepting the call. Such a challenge would have to be addressed by a practical context-aware call establishment application, but we assume, for our proof of concept, that enterprise users are trustworthy and use this application prudently.

3. Operational Description of CallerID++

Rather than providing a complete description of the operation of CallerID++, we demonstrate how CallerID++ would change the two scenarios outlined in Section 1 of this paper.

Scenario 1: When Bob calls Alice while she is on the conference call, CallerID++ detects that Alice is already on a phone call. Instead of establishing the call with Alice immediately, CallerID++ prompts Bob through his phone to answer a short series of questions, for example:

1. “State very briefly the purpose of your call”; Bob answers “Availability for our next project meeting.”

2. “What is the urgency of your call on a scale from 1-5 (5 being the highest)?” Bob answers “2”.
3. “What is the desired response time window if the callee cannot take your call (15 minutes, 1 hour, 1 day, 1 week)?” Bob answers “1 day”.

Depending on the context collection policy, CallerID++ may ask more, fewer, or different questions. The questions are presented as text-to-speech. The answers are collected either as an audio recording (question 1) or using a phone key-to-response mapping (questions 2, 3). CallerID++ now determines that the only endpoint that Alice is currently known to be present on is her desk phone and that her desk phone has no sophisticated text display capabilities beyond a simple LCD display. As a result, CallerID++ renders a signal tone in Alice’s leg of the conference call, followed by a header and a whispered message that includes Bob’s answers to the three questions above. Then, CallerID++ presents a whispered range of possible responses to Bob’s call request to Alice. Each response option maps to a key on her phone keypad. Based on the call context collected from Bob, Alice decides that it is sufficient to return Bob’s call some time during the day. By pressing (for example) “6” on her phone keypad, she selects *reject and auto-reconnect later*. Alice then turns her attention back to the voice conference. Finally, CallerID++ informs Bob that Alice has acknowledged his call attempt but is currently busy and that CallerID++ will later attempt to re-establish the call. Thus, both Alice and Bob offload the task of reconnecting later to CallerID++. Notice that embedding CallerID++ in the enterprise voice communication system in this scenario addresses Requirements 1-7 presented in Section 2. However, it is also clear that CallerID++ requires extra effort from Bob during the call context collection, which means that meeting Requirement 8 remains somewhat of a challenge. On the other hand, by keeping the number and complexity of questions to Bob low, we can improve the usability of CallerID++.

Scenario 2: At the time of Cynthia’s call to Alice, Alice has been chatting with Bob for a while. We may assume that there are no sensors in Alice’s office that monitor her activities away from her computer and communication devices and that CallerID++ has access to. Thus, Alice’s user context does not show any current or very recent known activity. In particular, Alice is currently not on the phone and is not known to be engaged in a very important activity. In such a situation, CallerID++ establishes a call right away without prior call context collection. Note that this is in congruence with Requirement 8 in section 2, i.e. it minimizes user effort during the call establishment process. However, CallerID++ does affect this scenario because it presents Alice with the entire spectrum of response options. Cynthia does not notice that CallerID++ is active but Alice does because she may choose a response option that is different from simply accepting or rejecting the call as in traditional telephony. She decides to press (for example) “3” on her phone for the *hold* response option. Her intention is to quickly wrap up her chat with Bob and then take Cynthia’s call. Cynthia hears a spoken message at her end of the call asking her to hold until Alice can speak to her. Then, Cynthia hears music on hold. Cynthia’s experience is very similar to that of calling into a call center and being placed on hold until the next available agent is found. Eventually, Alice politely finishes her chat with Bob and takes Cynthia’s call by pushing another button on her phone keypad. Notice that CallerID++ obviated the effort for Cynthia to leave a voice mail for Alice.

In both scenarios, CallerID++ increases the callee’s and the caller’s productivity, benefiting both the users and the enterprise. We feel that the overall advantages of CallerID++ are worth the extra effort that CallerID++ causes for the caller in Scenario 1. Moreover, while the caller may have spent more effort in Scenario 1 with the CallerID++ mechanism in the call path, there will be other scenarios where the caller would not have to leave voice mails for the callee or engage in phone tag. Thus the caller’s amortized extra effort with CallerID++ may well be negative, i.e. CallerID++ may save not only callees but also callers time and effort in the long run.

4. Architecture and Implementation of CallerID++

In this section we present a prototypical implementation of CallerID++. Although CallerID++ could be implemented without an enterprise communications middleware, our CallerID++ prototype uses the services of the *Hermes* [15] context-aware communications middleware at Avaya Labs Research.

We implemented our CallerID++ prototype around the Asterisk PBX [14]. Our choice of Asterisk as the PBX for this work is motivated by two reasons. First, as a PBX entirely implemented in software (for Voice over IP), it is easy to distribute, install, configure, and administer. Secondly, and for our purposes more importantly, a developer can easily extend and customize the Asterisk functionality with additional software modules that may be written in programming language of the developer’s choice.

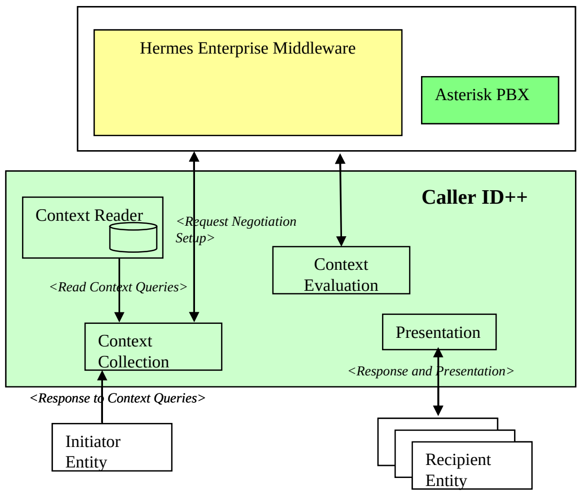

Figure 1: CallerID++ Architecture

Figure 1 shows the CallerID++ architecture. CallerID++ contains three major components, a *Context Reader, Context Evaluation and Presentation*, and *Context Collection*. From a high-level point of view, the responsibilities of these three modules are as follows. Depending on the callee's current user context and the caller's profile (see below), the call context may have to be collected from the caller. Gathering the call context translates into presenting the caller with a set of brief queries about the ensuing call, and, if he/she chooses to respond, collecting the user response to these queries. The *Context Reader* component dynamically populates the context queries required for call context collection, based on the callee's profile. *Context Collection*, on the other hand, sends these queries to the caller and constructs the call context based on the caller's responses. Subsequently, *Context Evaluation and Presentation* injects the call context into the existing call leg of the callee, to inform him/her of the specifics of the ensuing call, and presents the callee with several response alternatives. Depending on the callee's response, the new call is either established or not. The remainder of this section explains each of the modules, the design decisions, and the overall process in greater detail.

4.1. Choosing the appropriate query set

To reduce the effort and cognitive load on the caller during the call context collection stage, context queries are deliberately kept succinct and require only brief answers. The queries may contain, for example, (1) perceived importance, (2) the call topic, (3) desired response time window, and (4) expected duration of the ensuing call. Questions 1, 3, and 4 simply require a choice from a given set of possible responses. Additionally, rather than using static queries, we choose the context queries that are most suitable for the current call. For example, a set of queries based on the caller's enterprise profile and role is more appropriate for context collection than, say, using the same questionnaire for users across the entire enterprise. The role is a function of the caller and callee's positions in the organization and the topic of the call. The positions of caller and callee can be easily retrieved from the Hermes enterprise middleware, while CallerID++ has complete information of the topic of the call. In addition, the context queries can be changed easily to suit a specific enterprise or group need.

Selection of a query set can be considered as a policy defined by the enterprise or individual groups. The policy, maintained at CallerID++, denotes the most suitable query set for a specific type of calls and additionally for specific caller/callee combinations.

4.2. Triggering call context collection

During the call establishment, we could trigger call context collection in the following two ways: (1) during the establishment of every new call or (2) when the callee is busy with another call. While the first option may be able to provide useful information to the callee for each call he/she receives, providing context information for every call may be too cumbersome for the caller. Therefore, in our prototype, we activate the call context collection mechanism only if the callee is busy on another call. To determine whether the callee is busy on a call, we maintain a list of all active channels in the PBX and execute a simple search of the list when the caller places a new call. If the search concludes that the callee is on an existing call, the call is intercepted and the call leg to the callee is not established immediately. At this point in time, the callee continues with the ongoing phone call without any indication of a new call at her endpoint.

Next, we initiate the call context collection. Figure 2 shows the participating entities during call context collection. The *Context Collection* module runs as an Asterisk Gateway Interface (AGI) script, which facilitates interaction with the Asterisk PBX using standard interfaces. After intercepting a call, the *TextToSpeech* module converts the textual query set to audio files if the enterprise policy that defines context collection queries has changed. If not, the *ContextCollection* module retrieves stored audio files that correspond to the selected query set. Finally, the *UserInteraction* module uses standard interface operations to set up a dialog with the caller. The dialog contains the defined series of queries and collects the caller responses. Caller responses are stored on the Asterisk server as the call context. The *UserInteraction* module also provides error-checking during the dialog with the user by providing the user with an option to re-enter his response.

Extending this idea further, we want to query the context-aware communications middleware Hermes, for the full callee user context to determine whether the

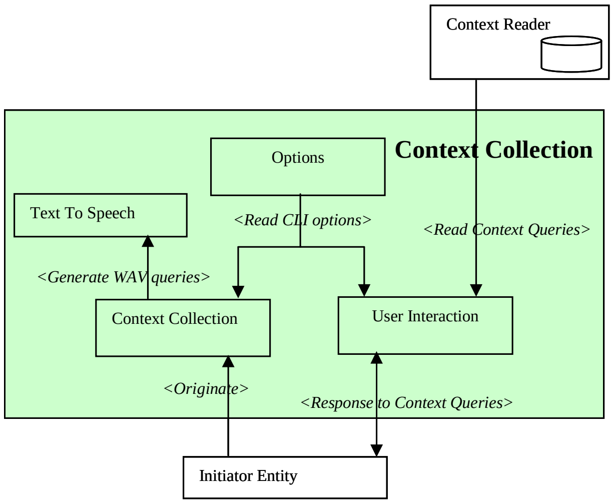

Figure 2: Call Context Collection

callee is engaged in other activities that make him/her potentially unavailable for this phone call. Interfacing with Hermes will also allow us to implement advanced functionality that depends on the caller's and callee's full user contexts. The reader is referred to [15, 16] for further details on the Hermes middleware platform, and its context collection mechanisms.

4.3. Rendering call context

The collected call context has to be presented to the callee who is currently active on another call. We use the *MeetMe* conferencing feature provided in Asterisk to render the call context. Figure 3 shows a sample call flow that illustrates the rendering process. In the call flow in Figure 3, we assume that users A and B are participating in an active call, and C originates a call to user A. Since A is busy on a call, CallerID++ collects the call context from C. Then, CallerID++ moves users A and B into a temporary *MeetMe* conferencing bridge. This call transfer is done transparently to users with no change in voice quality or user experience. Finally, the call context collected from C is conveyed to A as a low decibel "whisper" message in her voice channel. Thus, CallerID++ ensures the *continuity* of the current call i.e., it does not interfere with the callee's ongoing call. For the call context whisper, CallerID++ reuses the callee's currently active PBX channel and injects the context as a set of audio files.

In order to render the context information to other *types* of endpoints (cf. Requirement 5 in Section 2), for example an IM client or a Web browser, we have to first determine that the user is both present and available on such an endpoint, and then modify the context format for that endpoint. CallerID++ can query the Hermes middleware for other types of endpoints that the user is present and available on. However, the current prototype does not fully support context format transformation for rendering call contexts on non-voice endpoints.

4.4. Callee response

Once the call context has been rendered to A in Figure 3, CallerID++ asks A for a decision on how to respond to C's call attempt and provides a mechanism to collect A's response to this question. The response (whether the callee accepted the call or not) is conveyed to the caller implicitly, i.e. either the call is established (which conveys a *yes* from the callee) or call is terminated with a message indicating the reason (which conveys a *no* from the callee). Our existing callee response mechanism provides contact negotiation [13] in enterprise communications, which so far has been largely unaddressed. Moving ahead, we are interested in providing a broader range of response options to the callee, some of which are discussed in section 6.

4.5. User Experience

Since one of the most important goals for CallerID++ is to increase user productivity it should place the smallest possible cognitive load on its users.

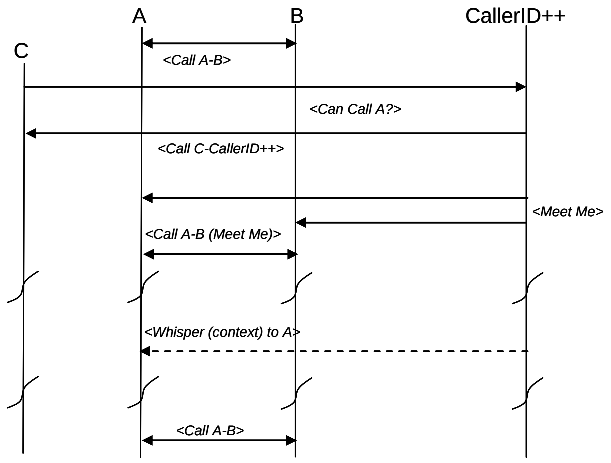

Figure 3: Call Context Presentation

During the execution of CallerID++, it interacts with two types of users, callers and callees. CallerID++

interacts with a caller initially for gathering call context and later to convey the callee's response to the caller's call request. For the callee, on the other hand, the only CallerID++ interaction is the whisper that gets injected into the ongoing call informing the callee of the ensuing call and the response options. Thus, during a single call establishment process, the total overhead CallerID++ places is proportional to the query set for the caller.

5. Related Work

Existing concepts that attempt to minimize or avoid unwanted interruptions of communication recipients and meet some or all of the other challenges for CallerID++ can be divided into two classes: some techniques provide the communication recipient with filtering methods that provide cues about the communications initiator and allow the *recipient* to decide whether to accept an incoming communication request [17]. Other techniques convey the recipient's context information to a communications initiator and allow the *initiator* to decide whether to make a communication attempt [12, 3, 1]. Neither class of techniques establishes a recipient's availability *relative to the specifics of the communication attempt*, which is the goal of CallerID++.

Lilsys [12] proposes the use of an array of physical sensors to gather context details of an enterprise user and then presents an inferred abstraction of the user's availability to a prospective caller. In addition, Lilsys provides mechanisms similar to Instant Messaging [5] for users to change their perceived availability status. The Awarenex [3] and Live Addressbook [1] systems on mobile devices provide communications initiators with location and availability status information about recipients. In contrast to these out-of-band user activity and context collection mechanisms, CallerID++ is essentially a transparent in-band context collection mechanism that gets triggered automatically only when the prospective recipient is busy (on a call).

Work discussed in [10, 11] exposes user context details to potential initiators at a much more fine grained level than Lilsys. However, a model that exposes detailed user information to an initiator may raise privacy concerns. Furthermore, it burdens the initiator with the task of interpreting the recipient's context details prior to every communications attempt, even if the recipient is currently available for every communication request. Forcing the initiator to make a communications decision may also have the undesirable side effect of unnecessarily deterring initiators from some communication attempts.

Simple filtering techniques, on the other hand, are often designed as rule-based, non-proactive systems. For example, such systems typically expect the user to define and change the *location* information i.e., the endpoint that user is currently associated with [1]. Furthermore, these techniques give little choice to the initiator during the communication establishment such as specifying the perceived urgency and the topic of the communication attempt.

Work in [8] proposes a talk time model based on a simple timer that allows enterprise users to agree on an appropriate time to talk. A graphical interface called *Negotiator* is deployed on a handheld device or a laptop computer so that users can exchange and possibly agree on the time of communication. CallerID++ and Negotiator both provide negotiation frameworks. However, the negotiation in CallerID++ is embedded into the communication path, while in Negotiator it is out-of-band. CallerID++ initiates negotiation on a communication attempt when the recipient is considered busy. Negotiator, on the other hand, begins negotiation prior to a communication attempt, regardless of the recipient's context. The basis for negotiation in CallerID++ is the call *intent* while in Negotiator it is the communication *time*. It would be interesting to combine the two technologies and measure the effectiveness of such a synergy.

SenSay [6] is a context-aware mobile phone that modifies its behavior based on its user's state and surroundings. Physical sensors placed on the user's body gather relevant data to determine the user's context. If a recipient is considered uninterruptible, any call attempt to the recipient's phone results in an SMS message that informs the caller of the unavailability of the callee. If the initiator chooses to try again after a certain configurable time period, the call goes through. CallerID++ is similar to SenSay in terms of collecting context data transparently. However, CallerID++ differs from SenSay in the following ways: It provides a call context collection mechanism that gathers call details such as the initiator's call intent and perceived call importance, it conveys the collected context to the recipient if the latter is busy (e.g., in another call), and it supports contact negotiation by allowing the recipient to choose from various responses to the call attempt.

6. Concluding Remarks and Future Work

In this paper, we showed that embedding context-aware applications in the communication paths between users in an enterprise environment leads to an increase in user productivity. Due to the pervasive nature of stationary and mobile enterprise telephony, we focused on voice communications and developed CallerID++, a context-aware application that improves upon the shortcomings of traditional caller ID. Unlike caller ID, CallerID++ collects the *call context* including the call purpose from the caller and makes it available to the callee to arrive at a better decision about call acceptance or rejection. CallerID++ also supports a variety of callee responses beyond call acceptance and rejection. We discussed the design and implementation of a CallerID++ prototype in conjunction with the Asterisk PBX.

We are currently implementing a variety of advanced response options available to a callee. Such options include *accept*, *reject*, *hold*, *reject and acknowledge, reject and auto-reconnect later*, *reject and request callback later*, *reject and offer callback later*. The *hold* option allows the callee to quickly wrap up an ongoing activity such as completing an email and to then take the call. While the caller waits for the callee, the caller might hear music on hold. The *reject and request callback later* option asks the caller to try calling again at a later time, whereas the *reject and offer callback* option would indicate to the caller that the callee will later try to establish the call. *Reject and auto-reconnect later* is an instruction to CallerID++ to reconnect the two parties at a time when both are likely to be available for the call.

Additionally, we intend to carry out usability studies of the CallerID++ application in an enterprise. For example, we are interested in determining the willingness of callers to answer the call context questionnaire. We also want to find the smallest size of the call context questionnaire that still conveys the salient points of the call context. Most importantly, we intend to measure the effective productivity increase through the use of CallerID++.

7. References

[1] A. E. Milewski, and T. M. Smith, “Providing presence cues to telephone users”, In Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW2000), NY, ACM Press, 2000.

[2] B. Nardi, S. Whittaker, and E. Bradner, “Interaction and Outeraction: Instant Messaging in Action. In Procedings of the Conference on Computer-Supported Cooperative Work (CSCW) 2000. NY: ACM Press, p. 79-88. 2000.

[3] J.C. Tang, N. Yankelovich, J. Begole, M. Van Kleek, F. Li, and J. Bhalodia, “ConNexus to Awarenex: Extending awareness to mobile users”, In Proceedings of the Conference on Computer Human Interaction (CHI), New York, 2001.

[4] J.J. Cadiz, A. Narin, G. Jancke, A. Gupta, and M. Boyle, “Exploring PC-telephone convergence with the enhanced telephony prototype”, In Proceedings of the Conference on Computer Human Interaction (CHI), New York, USA, 2004.

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

[6] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, and F. Lung Wong, "SenSay: a context-aware mobile phone", In The Proceedings of International Symposium on Wearable Computers (ISWC 2003) 2003, White Plains, NY.

[7] Microsoft Office Live Communications Server, <http://office.microsoft.com/en-us/communicationsserver/default.aspx>, on 02-15-2007.

[8] M. Wiberg, and S. Whittaker, “Managing Availability: Supporting Lightweight Negotiations to Handle Interruptions”, ACM Transactions on Computer-Human Interaction, Vol. 12, No. 4, December 2005.

[9] J. M. Hudson, J. Christensen, W. A. Kellogg, and T. Erickson, “I’d Be Overwhelmed, But It’s Just One More Thing to Do: Availability and Interruption in Research Management”, In Proceedings of ACM Conference on Human Factors in Computing Systems (CHI2002), New York: ACM 2002.

[10] M. Danninger, T. Kluge, and R. Stiefelhagen, "MyConnector – Analysis of Context Cues to Predict Human Availability for Communication", In The Proceedings of the International Conference on Multimodal Interfaces (ICMI 2006), Alberta, Canada November 2-4, 2006.

[11] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, "ContextPhone - A prototyping platform for context-aware mobile applications", IEEE Pervasive Computing, 4 (2): 51-59, 2005.

[12] J. Begole, N. E. Matsakis, and J. C. Tang, "Lilsys: Sensing Unavailability", In the Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004), Chicago, Illinois, November 6-10, 2004.

[13] J. C. Tang, “Approaching and leave-taking: Negotiating contact in computer-mediated communication”, ACM Transactions on Computer-Human Interactions (ToCHI), 2001.

[14] Asterisk open-source PBX and telephony toolkit, <http://www.asterisk.org/>, on 02-15-2007.

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

[16] A. Kavimandan, R. Klemm, A. John, D. Seligmann, and A. Gokhale, "A Client-Side Architecture for Supporting Pervasive Enterprise Communications”, Proceedings of The IEEE International Conference on Pervasive Services (ICPS 2006), Lyon, France, June 26-29, 2006.

[17] E. Horvitz, A. Jacobs, and D. Hovel, “Attention-sensitive Alerting”, In Proceedings of Conference on Uncertainty and Articial Intelligence (UAI 1999). 1999. Stockholm, Sweden: Morgan Kaufmann. p. 305-313.