Context-Aware Unified Communication

Hui Lei
IBM T.J. Watson Research Center
hlei@us.ibm.com

Anand Ranganathan[*]
University of Illinois at Urbana-Champaign
ranganat@uiuc.edu

Abstract

Enabling people-to-people interaction across heterogeneous communication end-points enhances user experience and fosters people collaboration. This paper presents the design and implementation of a unified communication system, dubbed Mercury, that allows a user to interact with others using the most convenient device at the time. Mercury supports both two-way conversation and one-way messaging. It leverages the Session Initiation Protocol to manage communication sessions and exploits dynamic user context to proactively route and migrate calls. It allows for subscription to other users' unified reachability status and provides a soft ring feature via universal notification. Mercury has an extensible architecture that allows new device types to be easily incorporated into the system. Our prototype implementation integrates a variety of devices: telephones, Sametime instant messaging clients, email, and pagers.

1. Introduction

Modern man is part of a highly connected communication network. People can interact with each other through a wide variety of communication mechanisms, such as email, instant messaging, cellular phone, landline phone, SMS, voice-mail, and pager. Each means of communication has its own sets of features and drawbacks. Although a person typically has multiple communication devices[1], he may have access to only a subset of them at a particular time. Depending on his situation, he may also have a preference on which of the available devices to use. For example, a person may prefer chatting with somebody using an instant messaging (IM) client when he is working on something else or in the middle of a meeting. But when the meeting is over or if he has to leave the room, he may want to continue chatting with the other party on his cell phone or via short messaging service (SMS). Hence, a unified communication system that allows a person to communicate using the most convenient device at the time will enhance user experience and offer more opportunities for collaboration.

People-to-people interaction falls into one of two categories: two-way communication (i.e., conversation) or one-way messaging (i.e., notification). Two-way communication is inherently more complicated than one-way messaging. First, messaging is asynchronous in that there may be arbitrary time lapse between the sending and receiving of a message. In comparison, conversation is synchronous: both parties must be present in order for a conversation to take place. Thus, conversation requires proper call setup. Call setup alerts the callee and obtains her acceptance for the call. It further involves negotiation between the devices on what media and data formats should be used for communication. Second, while messaging is stateless, conversation consists of a sequence of exchanges and is stateful. Call state must be maintained for the entire duration of the call. Also, call migration from one device to another may be desirable and needs to be supported. Despite the differences, messaging can be effectively treated as a special case of conversation, where the conversation consists of just one message.

A promising technology for unified communication is the Session Initiation Protocol (SIP) [1]. SIP is an application-layer control or signaling protocol for creating, modifying and terminating sessions with two or more participants. These sessions include Internet multimedia conferences, Internet telephone calls, multimedia distribution, and instant messaging. However, SIP only provides a mechanism for managing calls. It does not specify what policies should be used for call management or how the policies should be enforced. It is obviously impractical to expect users to manually and constantly control all the call aspects such as where to route a call and whether to migrate a call. Further, many legacy systems and devices are not yet SIP-enabled and therefore may not be directly plugged into the SIP framework.

[*] This work was performed while the author was visiting IBM Watson Research

[1] In this article, we use the word "device" in a broad sense to mean either hardware entities (like phones, pagers, etc.) or software entities (like instant messaging clients, email clients, etc.)

We have designed and implemented a unified communication system, dubbed Mercury[2]. While using SIP as the underlying mechanism for creating, maintaining and terminating calls, Mercury exploits dynamic user context information (e.g., user location and activity) to proactively manage communication sessions. Context awareness [9] is a well-recognized technique in pervasive computing. It enables personalized access to services and reduces demand for user attention. Not only does Mercury use context information to decide upon the most appropriate device to which an incoming call should be routed, it also monitors the context of the communicating parties and decides if the call should be migrated to another device due to change in user context. In addition, Mercury uses context information in a number of other value-added features to enhance the overall user experience.

To illustrate the main capabilities of Mercury, we present a scenario that shows Mercury in action.

Bob is having a meeting in his office. His preference profile in Mercury specifies that, during meetings, he will not take calls from his friends, but he will answer calls from his family and using his IM client. Now a friend, Alice, wants to communicate with Bob. She checks his unified reachability status from her SMS device and finds that he is not available to talk to her at this time. Therefore she submits to Mercury a subscription for Bob's unified reachability status so that she can be notified when Bob's availability changes. A few moments later, Bob's wife, Carol, calls him from a telephone. Mercury notices that Bob prefers to use the IM client for two-way communication under the circumstance. However, Bob is not currently logged on the IM system. The only device he is connected to is his SMS-enabled cell phone which has been set on vibration mode. So Mercury uses the soft ring feature to notify Bob through SMS that Carol wants to get in touch with him. Upon receiving the notification, Bob immediately logs on the IM system, which leads Mercury to route Carol's pending call to Bob's IM client. Bob and Carol thus have a conversation with Bob using his IM client and Carol using her phone. When the meeting is over, Mercury detects the context change and asks Bob whether he would like to use his office phone instead. With Bob's confirmation, Mercury transfers the call to Bob's office phone without Carol being aware of the change. After Bob finishes talking to Carol, Mercury sends a callback to Alice informing her that Bob is available to talk to her. So Alice calls Bob from her SMS device. Although Bob is reachable on IM client, cell phone as well as office phone, Mercury properly routes the call to Bob's office phone, according to his preference.

[2] Our system is named after the light-footed messenger God of Roman Mythology.

As exemplified by the above scenario, Mercury has three novel features:

- Proactive call routing and call migration based on dynamic user context information.
- Support for the querying and notification of a user's unified reachability status, which considers the user's availability across all of his devices.
- Soft ring: using an appropriate messaging device to alert the callee of an incoming two-way call.

The Mercury features embody two main thrusts: integration and context sensitivity. While prior work is limited to device integration in the initial routing of communication [2, 3, 4], Mercury goes far beyond that. It also integrates devices when reporting a user's reachability, when ringing the callee, and when transferring an on-going call. In addition, the policies that drive all the integration features in Mercury are expressed in terms of user context, which makes the integration more adaptive and flexible. Architecturally, Mercury demonstrates how all the above features can be implemented in a standard SIP framework. Further, Mercury makes use of an infrastructure context service to gather and manage a wide array of user context information. This simplifies the core logic for communication management. It also allows Mercury to easily broaden its use of context information as the latter become available via the context service.

The rest of the paper is organized as follows. Section 2 presents considerations underlying the Mercury design. Section 3 describes the actual design of the architectural components. Section 4 describes in detail various system operations. Section 5 discusses the prototype implementation and our experiences. Section 6 discusses related work and Section 7 summarizes our conclusions.

2. Design Rationale

A number of considerations have influenced the design of our system. Among them are context awareness, device heterogeneity, user privacy, reachability status, and system extensibility. In this section, we elaborate on these issues.

A critical issue in any unified communication system is the selection of an appropriate device for the recipient. For a conversation session, the choice of the appropriate device may also change during the session, for either the caller or the callee. In today's communication systems, selection of communication devices is left to the caller. The callee is forced to use whatever device the caller chooses for the entire conversation, even though it might not be the most appropriate device for his or her context. Also, the caller may not know how and where to reach the callee. Consider the use of telephones: the caller is responsible for deciding which phone number to dial (he has to choose from multiple home, office and mobile phone numbers). Also, the callee is now forced to talk on the phone, even though he may currently be in a meeting or talking to somebody else. Besides, the end-users are responsible for manually tranferring an on-going call to another number, if it is at all feasible.

Device selection can be expected to be much more burdensome when multiple, heterogeneous devices may be associated with a user. For instance, sometimes people may have to instant-message each other to find out whether it is a good time to talk on the phone and what phone number should be used. We advocate exploiting user context information to proactively select the communication device. The best means of reaching a person at a particular time depends on many factors, including the person's location, activity and connectivity. Such attributes are often referred to as user context. By obtaining information on dynamic user context, the system can make call routing and migration decisions automatically, based on a user preference profile. This reduces the demand for user attention and potentially improves user productivity. On the other hand, there is a limit to the use of context data. Sensed context may be ambiguous or incomplete, and user intention may not be perfectly inferable. Therefore, a user should also be allowed to override the system's decision by providing explicit input.

Integrating heterogeneous devices in a unified communication system supports personal mobility and promotes people-to-people reachability [2, 3]. Devices may differ in communication protocols, data modalities and formats, and even duplex mode. Depending on the circumstance, some types of communication devices may be more preferable than the others. In particular, messaging devices such as pagers, email and voice mail do not support conversation sessions between users. Nevertheless, they are very useful for sending messages and are popular communication mechanisms. It is thus important to provide an overarching communication framework that integrates all types of devices, including messaging devices. An added advantage of such integration is that messaging devices can even be used to enhance conversation. There may be times when a callee is accessible via some messaging devices only, but not via any conversation devices. The system may choose to send a notification to a proper messaging device informing the user of the in-coming call, who can then take the call on a conversation device. This is the soft ring feature we alluded to earlier.

Unified communication should be provided without compromising the privacy of the parties involved [2]. Each party should control which device to use for a particular communication. Due to their private nature, user context information and preferences that are used for device selection should not be revealed to other parties. Even the device of choice should not be disclosed because it could give a clue on the user's activity and location. Further, receiving unwanted calls is also an intrusion of privacy. Thus the system must allow for prioritization and filtering of calls based on user preferences.

The fact that conversation is synchronous imposes requirements on the connectivity of both parties. Information on a person's reachability status helps others to decide whether it is possible to place a call. Reachability status is already available in some homogeneous communication systems. For example, chat programs indicate whether the other party is online and can be reached for a chat session. Since Mercury proactively selects communication devices for users, it is natural for the system to present a unified picture of people's reachability. Mercury integrates all the presence information from different devices and is able to tell users if a particular person is reachable on any communication device. Unified reachability status simplifies the task of potential callers and in the meantime better preserves the privacy of the callee A

Last but not least, extensibility is key to any unified communication framework [3]. We have seen a proliferation of communication capabilities and devices, and this trend is likely to continue. A unified communication system, intended to accommodate heterogeneous device types, should allow for evolution of existing communication capabilities and integration of new capabilities. Supporting upgraded or emerging devices should involve minimal development and deployment effort.

3. System Design

In this section, we explore the architecture of the Mercury communication system. We describe the user interface as well as different components of the system.

3.1. Architecture

Figure 1 shows the overall system architecture. The Mercury system comprises an extensible set of *Device*

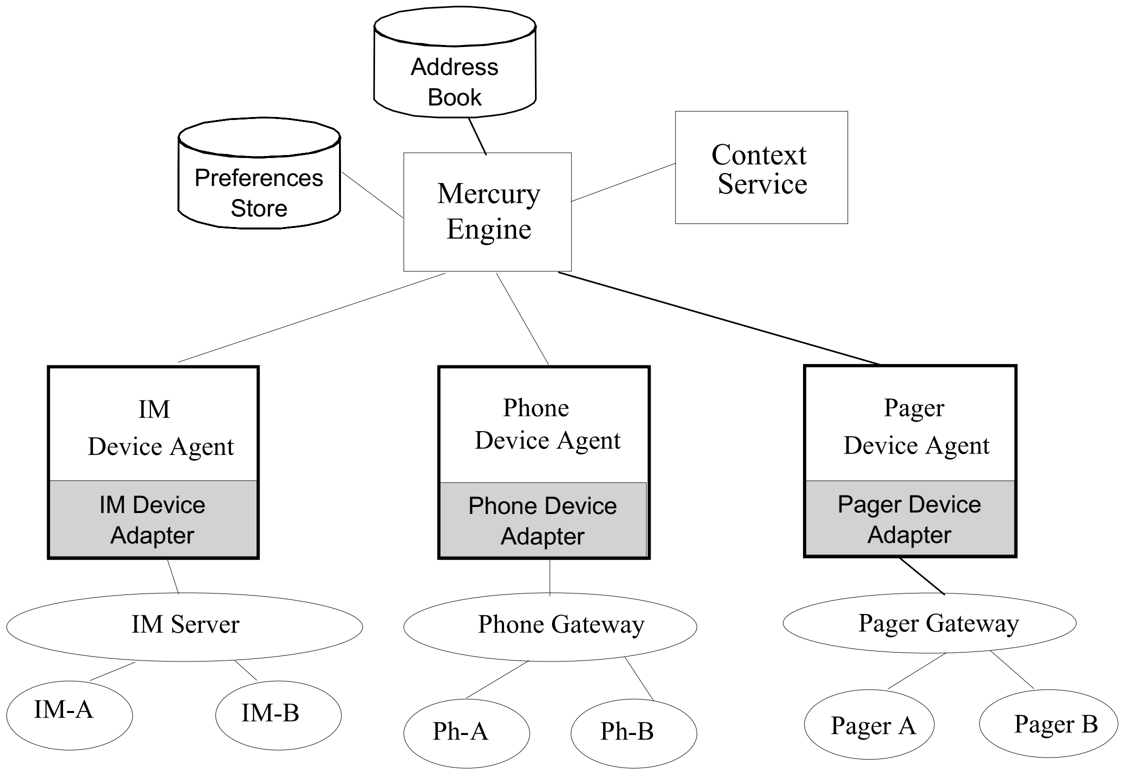

Figure 1. Mercury Architecture

Agents, the *Mercury Engine* and the *Context Service*. Each Device Agent serves as an access point for one type of communication devices and manages sessions involving those devices. It includes a device-dependent adapter that encapsulates the details of interaction with devices of a particular type. Both conversation devices (such as IM clients, SMS devices and phones) and messaging devices (such as pagers and email clients) may be integrated into Mercury by means of Device Agents. The Mercury Engine performs address lookup using an address book. It also makes call routing and migration decisions based on user preferences and user context information that is provided by the Context Service.

Mercury uses SIP as the underlying mechanism for creating, maintaining and terminating communication sessions. In fact, Device Agents and the Mercury Engine are all SIP entities: the former are SIP end points and the latter acts as a SIP server. They exchange SIP messages in order to manage calls.

3.2. User Interface

Mercury allows a user to perform the following functions:

- *Place a call.* The callee is identified by either a globally unique ID (GUID) or a device-specific address. The use of device-specific address is simply for the caller's convenience and does not mandate the use of that device. For example, if the caller is using a telephone, it may be easier to enter the callee's telephone number instead of some other alphanumeric ID.
- *Transfer a call.* Either party in a call can switch from using one device to another without disrupting the flow of conversation.
- *Terminate a call.* Either party in a call may close the communication session at any time.
- *Send a message.* A user may send a message to another person. Both messaging and conversation devices of the receiver would be considered for delivering the message.
- *Specify reachability.* A user can indicate what devices he may be reached at by marking one or more of his devices as active or inactive.
- *Request for unified reachability information.* Users can query about a particular person's unified reachability status or, when the person is not currently reachable, subscribe for her

reachability status so that they can be notified when the person is reachable. The reachability status is a function of the requester identity and the dynamic context of the person in question. A person that is physically connected to some device is considered unreachable if his preference profile prevents Mercury from routing incoming communication to him.

It should be noted that Mercury is able to make call migration decisions and determine users' reachability automatically. Still, it allows explicit user input to complement automatic control.

A user may interact with Mercury using any communication device. The user enters commands through the device's native interface. The commands are then transferred to the Device Agent for the device type, which has a well-known address on the access network. For example, the IM Device Agent is another user in the instant messaging system. The user may start an IM session with the Device Agent and send it various commands. Similarly, the phone Device Agent may be reached at a standard telephone number. The user dials this number to use Mercury.

Conversation messages with the other party are also entered through the same device interface and mediated by the appropriate Device Agent. Because conversation messages and control messages may inter-mix, they need to be distinguishable from each other. Separation of call data and call control is handled in a device-specific way. For instant messaging, all user commands are prefixed by a "$". For telephones, user commands are entered by pressing various keys and communication messages are via voice.

3.3. Device Agents

Device Agents allow disparate devices to be integrated into Mercury. They are addressable SIP entities and are capable of originating and terminating SIP requests. Each Device Agent handles one type of communication devices and acts as an access point for those devices.

A Device Agent performs three kinds of functions. First, it interacts with devices of a particular type. The Device Agent initiates and terminates calls on the devices. It accepts control and conversation messages from the devices, and sends response messages to the devices. Second, the Device Agent implements a SIP user agent. It constructs SIP messages (including presence messages in extended SIP [6]) and sends them to SIP entities such as the Mercury Engine and other Device Agents. It also listens for various SIP-related messages and events. Third, the Device Agent relays conversation messages to and from other Device Agents. If necessary, it also translates those messages into different modalities or languages.

A Device Agent consists of a device-independent component, called the *Agent Core*, and a device-specific component, i.e., the *Device Adapter*. The Agent Core handles interaction with the Mercury Engine and other Device Agents, whereas the Device Adapter handles interaction with devices. The interaction between the Agent Core and the Device Adapter is through standard interfaces. Specifically, Device Adapters across all Device Agents implement a uniform adapter interface so that the Agent Core components may interact with them in a device-neutral manner. Another programmatic interface abstracts the user-related functionality of the Device Agent, to which the Device Adapter maps user input.

3.4. Mercury Engine

The Mercury Engine is essentially a SIP server. It forwards call requests to appropriate Device Agents. It monitors user context during a call and, if necessary, prompts the user to transfer the call to another device. It accesses an address book to map between a user's GUID and various device-specific addresses. In addition, the Engine accepts registration of and subscription for presence information, and sends notification of reachability. The Engine's presence capability builds upon the functionality of the external Context Service.

The Mercury Engine makes call routing and migration decisions based on individual users' preferences. A user's preferences are expressed as a set of rules. Each rule specifies the devices that may be used under a particular condition. The rule condition is in terms of the callee's context variables (e.g., location, activity) and/or the attributes of the caller (e.g, caller ID, caller group). Each rule is optionally associated with a priority value to help resolving conflicts between rules.

Although we talk about the Engine as a single logical unit, the Engine functionality may be physically replicated so that each Engine instance services only a subset of the users. For example, an Engine instance may be deployed for one administrative domain or, in the extreme case, for a single user. This way, the Engine instance is exposed only to the preferences and context information of the users it services, resulting in better security and privacy. In addition, since the service load is divided among multiple Engine instances, the system can scale better.

3.5. Context Service

The Context Service, described in detailed in a separate publication [4], allows the Mercury Engine to obtain user context information without having to worry about the details of context derivation and context management. The Context Service API includes both synchronous query and asynchronous callback functions. It is also very easy to incorporate new types of context data into the Context Service. Information currently provided by the Context Service includes IM online status, activities and contact means derived from calendar entries, desktop activities, as well as user location reported from a variety of sources such as cellular providers, wireless LANs, GPS devices, and RIM blackberry devices.

In addition to providing user context information, the Context Service also provides the basis for the presence capability in Mercury. In fact, reachability state is treated as one type of context information and maintained by the Context Service. The context update and callback functions in the Context Service directly correspond to the REGISTER, SUBSCRIBE and NOTIFY features in SIP.

4. System Operations

In this section, we describe how various system operations are performed.

4.1. Call Creation

The successful creation of a call is shown in Figure 2. Let us assume that person X wants to converse with a person Y. So, X would pick up any device that is most suitable at the time. For example, he may use an IM client if he is next to a computer or he may use his cell phone if he is on the road. The user calls the Device Agent first and, using the native device interface, asks the Device Agent to start a session with Y **(Step 1)**. The Device Agent then sends a SIP INVITE request to the Mercury Engine indicating the address of the other party **(Step 2)**. The Engine finds the GUID of Y, if necessary, by referring to the address book, looks up the preferences of Y and obtains the current context from the Context Service **(Step 3).** The types of context information that can be used in the preferences and in making call routing decisions include location of the user, activity, status on an instant messenger and time of day. The Engine then sends a SIP INVITE to the appropriate Device Agent **(Step 4)**. The Device Agent receiving the INVITE indicates to Y, through the Device Adapter, that he has an incoming call from X **(Step 5)** and allows him to accept or reject the session **(Step 6)**. If Y accepts the session, a positive response is sent back to X via the Engine and X's Device Agent **(Steps 7, 8, 9)**. The caller's Device Agent then sends back an ACK to the callee, completing the 3-way handshake for creating a session **(Steps 10, 11)**. The callee's Device Agent then opens a socket to the caller's Device Agent and conversation messages are exchanged between the caller and the callee via the socket **(Step 12)**. If the callee rejects the session, a negative response is sent back to the caller's Device Agent via the Engine. The caller's Device Agent asks the caller whether she would like to leave a message. If the caller so chooses, the Device Agent then starts a session with one of callee's messaging devices. The operation of messaging is described in Subsection 3.4.

The INVITE request also indicates the data types (like text, audio, etc.) the caller's Device Agent is able to support. The callee indicates the data type it prefers to receive in its response to the caller. If the callee cannot understand any of the caller's data types, it sends back a negative response, indicating what data types it supports. The caller can then re-send the INVITE request if it is able to support any of the callee's data types.

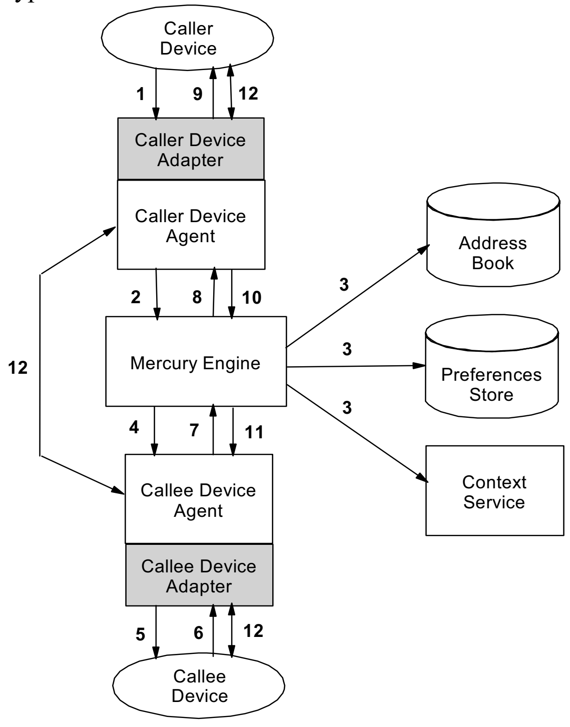

Figure 2. Call Creation

4.2. Call Migration

The most appropriate device for a user may change during a call. For example, a person who uses a portable SMS device while walking to his office may want to switch the conversation to a desktop IM client once he is in the office. Mercury monitors a user's context and proactively prompts the user to switch to a more convenient device. The call flow for such a proactive call migration is shown in Figure 3.

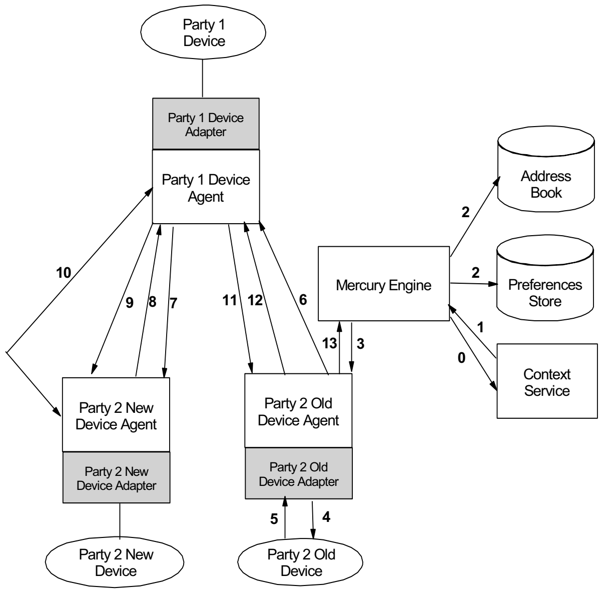

Figure 3. Call Migration

When a call involving some user is first created, the Mercury Engine creates subscription with the Context Service for changes in the user's context **(Step 0)**. When changes occur, the Context Service sends the Engine a callback **(Step 1)**. The Engine checks the user's preferences to determine whether the context change warrants a switch of user device and, if so, looks up the address of the new device to which the call should be transferred **(Step 2)**. The Engine then sends to the old Device Agent of the user a NOTIFY message containing the address of the new device **(Step 3)**. The user is then asked on the device he is using if he wants to migrate the call to the new device **(Step 4)**. If the user accepts the transfer **(Step 5)**, the Device Agent then sends a REFER message to the other party **(Step 6).** The REFER request is a standard SIP message for transferring calls. It instructs the receiver to start a new session with the address referred to. Once the other party gets the REFER request, it sends an INVITE to the new device and starts a session with the new device using the standard SIP 3-way handshake **(Steps 7-10)**. Once the new session is set up, the old session is terminated by the other party's Device Agent sending a BYE message to the user's old Device Agent **(Step 11)**, which then replies with a positive response **(Step 12)**. The old Device Agent also sends a NOTIFY to the Engine informing it of the successful call transfer **(Step 13)**.

Call migration may also be initiated by the user specifying a new device explicitly. Manual call migration works in a similar way as a proactive migration. It follows Steps 5-13 in Figure 3.

In order for the Mercury Engine to proactively recommend call migration, it must be aware of the state of each call. That is why the Device Agent sends it a Notify message after a successful call migration (Step 13 above) and after the call is terminated (see next subsection).

4.3. Call Termination

To terminate a call, the user indicates to Mercury that he wants to close the session. The Device Agent then sends a BYE request to the other party. The other party sends a positive response to the BYE and the session is closed. The Device Agent also sends a NOTIFY to the Mercury Engine indicating that the call is terminated, so that the Mercury Engine is aware of the current state of the call.

4.4. Messaging

A user can send a one-way message to another user's device. Messaging is treated as a special case of synchronous conversation. The choice of which device to use is again made based on the context and the preferences of the intended receiver. A session is created between the sender's Device Agent and the receiver's device, in the same way as for two-way conversation. The Agents at either end can once again negotiate the data format of the messages. The only difference is that the recipient's Device Adapter must buffer messages from the sender until the session is terminated by the sender. This is necessary because the sender may be using a conversation device (e.g., IM client) and thus may send multiple messages, while the receiver may be using a messaging device (e.g., email). Once the sender terminates the session, the Device Adapter of the receiving device sends a single message to the intended receiver containing all messages from the sender.

4.5. Soft Ring

It is possible that when a two-way call request arrives at the Mercury Engine, the callee is either not reachable on any conversation device, or the current context and his preferences dictate that the caller cannot reach him on a conversation device. For example, he may be away from his office phone, and is not running the IM client. However, if he is still reachable through some messaging devices, the soft ring feature of Mercury notifies the callee of the incoming call via an appropriate messaging device.

If the callee desires to start a conversation with the caller, he makes himself available on a conversation device. For instance, he can log into his IM client, or go to his office, or supply an alternative phone number to Mercury. Mercury then redirects the call to this conversation device.

The call flow for soft ring is shown in Figure 4. Once the Mercury Engine gets an INVITE request from a caller **(Steps 1-2)**, it looks at the preferences and the context of the callee to select an appropriate conversation device **(Step 3)**. If no conversation device is available but the use of a messaging device is allowed by the user preferences, the Engine sends a MESSAGE request [8] to the appropriate Device Agent **(Step 4)**. This MESSAGE request contains information on the incoming call. The Device Agent in turn sends a notification to the user's messaging device **(Step 5)**. The Engine then subscribes with the Context Service for the callee's connectivity through a conversation device **(Step 6)**. The subscription has an expiration time. When the callee becomes available on one of the conversation devices, the Engine is notified **(Step 7)**. The Engine then forwards the original INIVTE request to the conversation device **(Step 8)** and the session is set up between the caller and the callee as before **(Steps 9-16)**. All these actions taking place at the callee's end is not visible to the caller. The caller just sees that the session is set up finally. If the callee does not make himself available within the Engine timeout period, the Engine just sends a negative response to the caller saying that the callee could not be reached.

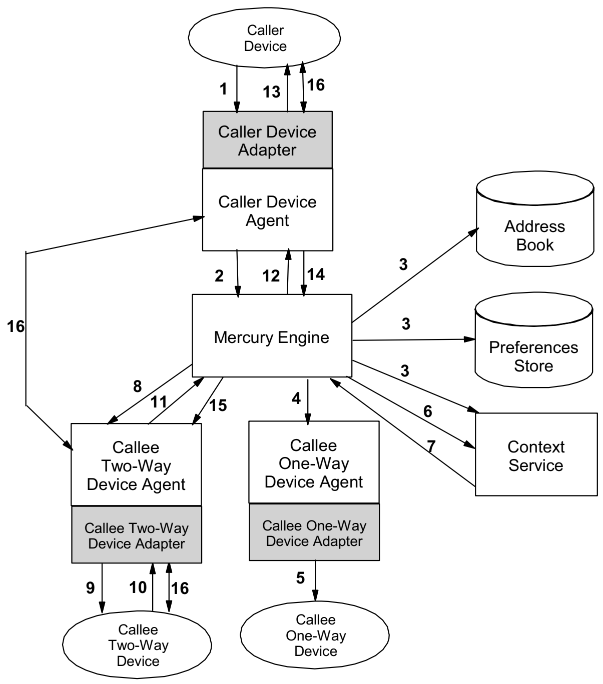

Figure 2. Soft Ring

4.6. Unified Reachability Status

Mercury reports a person's unified reachability status with respect to a particular requester. To achieve that, Mercury takes into account the following information of the subject: connectivity state on various devices, context attributes such as location, activity and whether the person is currently in a call, preferences applicable to the current context and the requester. The connectivity state, like other context attributes, is obtained from the Context Service. The connectivity state may be either automatically sensed or manually asserted.

When a user explicitly specifies her connectivity on a device through a Device Agent, the Device Agent forwards the information to the Mercury Engine via a REGISTER message. The Engine then pushes the information to the Context Service as a connectivity update. Similarly, a subscription for unified reachability status is forwarded from the Device Agent to the Engine via a SUBSCRIBE message; the Engine then enters a callback request to the Context Service on change in the subject person's context. When the Context Service later issues a callback to the Engine, the latter determines whether the context change results in the subject becoming available. If so, the Engine notifies the original subscriber using the unified messaging scheme outlined in Subsection 4.4.

The use of the Context Service for reachability status offers a number of advantages. First, the built-in support for context publication and subscription in the Context Service simplifies the logic of the Mercury Engine. Second, the Context Service is able to sense different types of context information like location, and then automatically infer that the user is reachable on certain devices (like telephones in the room he is in). The Context Service is able to fuse potentially conflicting context data from multiple sources [18]. This allows user-asserted presence to be fused with automatically sensed connectivity, providing aggregated connectivity state and with better quality. Finally, the Context Service has mechanisms for managing the privacy of users. It uses Role-Based Access Control (RBAC) to determine if a certain requestor is granted access to reachability information about a particular subject. The Context Service allows users to specify access control policies that guard information about their reachability on any device.

5. Implementation and Experiences

Our current prototype implementation integrates a number of disparate devices - telephones, Sametime instant messaging [10], email and paging. The implementation was done primarily in Java.

Specifically, we use IBM's DirectTalk [12] to interface with the telephone network, the client toolkit that comes with Sametime to send and receive instant messages, and the messaging services and push toolkit provided in IBM's Websphere Everyplace Access [13] to connect to email clients and pagers. In addition, speech-to-text and text-to-speech conversions are performed using IBM's ViaVoice software [14], and SIP functionality is accessed via the JAIN SIP API [15].

Our prototype of the Mercury system performed well in enabling communication between different types of devices. Communication between even the most disparate devices (instant messaging and telephone) was smooth. There was no noticeable extra time-lag between typing a message in the instant messenger and hearing it on the telephone at the other end. In fact, to the person using the instant messenger, there wasn't much difference between communicating with another instant messenger and with a telephone. The only exception to this finding was that it was not possible to use the standard acronyms and emoticons, which have become an integral part of instant messaging conversations today. This problem can be overcome by performing transcoding like expanding acronyms or using emoticons to change the tone of the voice. To the user on the telephone, too, the conversation flow was smooth, apart from the fact that the voice of the text-to-speech software was the same for all users. This reduces the personal touch that characterizes telephone conversations. While there was no problem with text-to-speech, the speech recognition only worked well with a limited vocabulary. This is an intrinsic limitation with the current speech recognition technology. We are hopeful that, with improvements in speech recognition software, our system will perform more seamlessly.

Since communication devices of today are not SIP enabled, users had to first establish sessions with the SIP-aware Device Agent before they could use any of the features of Mercury. For instance, in the case of instant messaging, they first had to start a session with the IM Device Agent (which would be present in the buddy list). For telephones, they first had to dial the number of the Phone Device Agent. Once they were in a session with the Device Agent, they could make calls and perform other call control actions like transfer and checking the reachability of others. This extra step of creating a session with the Device Agent could be avoided if the device is SIP-enabled. In the future, we foresee that many IM clients would use SIP. Also, IP telephones that use SIP would become common. In such a scenario, users can directly use Mercury's features from their devices (i.e. their devices would be able to perform all the functions of the Device Agent).

Once the user is in a session with the Device Agent, he can take advantage of the wide variety of features that Mercury offers. We have tried to make the user interface intuitive and similar to existing interfaces. For example, when the user is on a telephone, he is first presented with a voice-based menu of options that include creating a call, checking the reachability of others and modifying privacy preferences for the phone. He can choose an option by pressing the appropriate key. If he is in a conversation with somebody else, he can place the call on hold at any time and access a menu of options by pressing the "#" key. This menu of options also allows him to transfer the call to another device. Similarly, when the user is using the IM, he can perform various actions by sending control messages to the IM Device Agent. Control messages are distinguished from normal conversation messages by the fact that they are prefixed by a "$".

In our current prototype, users can specify preferences for incoming calls in a preference profile, which is written as a text file. Users can specify preferences that specify which communication device should be used in different contexts and for different callers. This text file is then used by the Mercury Engine for routing calls to the appropriate device in different contexts. We are working on a more intuitive graphical interface for users to specify these preferences.

The process of migrating calls was also fairly smooth. Users could manually migrate calls to other devices at any point in the conversation. Automatic call migration, wherein the system proactively migrates the call to a more suitable device was also useful in some situations. To avoid startling the user by migrating calls unexpectedly, the Mercury system prompts the user and asks for her approval before performing the call migration.

The use of SIP standards by Mercury allows it to inter-operate with other SIP-based systems. Because the key functionalities of Mercury, such as context-aware call control, soft ring, and unified reachability status, are all implemented as services (i.e., Mercury Engine and Context Service) in the core infrastructure, new SIP endpoints may be integrated into our system easily. Further, the innovative features of Mercury will become immediately available to these new devices.

6. Related Work

A number of projects have addressed the issue of personal mobility to support unified communication. The Mobile People Architecture (MPA) introduces a person layer on top of the application layer to emphasize that the person, rather than the device, is the communication endpoint [2]. A person-level router, Personal Proxy, tracks a person's connectivity, accepts communications on his behalf, converts them into different application formats, and forwards the resulting communications to him. Routing decisions are based on a variety of information, including the user's connectivity state, communication metadata like sender and date, and communication content. Obviously, it is not feasible to rely on communication content for routing decisions in two-way communication, which is not available during the initial call setup. In fact, it is not clear how two-way communication can be handled by MPA, as there does not seem to be a signaling protocol.

Universal Inbox enables redirection of in-coming communication based on user preference profiles [3]. Built on the ICEBERG architecture [19], Universal Inbox identifies three distinct requirements of unified communication and realizes them as reusable infrastructure services: data transformation, preference-based redirection, and name mapping. It uses a home-grown signaling protocol and supports manual, but not proactive, call migration. Although it was suggested in [3] that redirection preferences might be predicated on dynamic user context such as location and state, the feature did not appear to have been implemented.

Mercury shares some design goals as MPA and Universal Inbox. These include respecting user privacy, giving control to the callee, and system extensibility. While the other two projects support integrated in-coming communication, Mercury identifies additional aspects where an integrated service can be useful, ranging from call migration to reachability status and to call announcement. Mercury implements the integration features using a well known protocol – SIP, and uses a much broader array of user context to direct different types of integration. It demonstrates how SIP can interoperate with an infrastructure context service.

Certain aspects of Mercury have been explored in recent research efforts. In [11], Singh and Schulzrinne discuss the use of SIP in a multimedia mail system to solve call forwarding, reclaiming and retrieval of messages. The Family Intercom [17] system aims to facilitate audio communication in a domestic environment. It uses the context toolkit to gather location and activity information of the callee. It then communicates such information to the caller for him to decide whether it is socially appropriate to initiate a conversation. Recognizing the importance of context information in communication, ConNexus and its successor, Awarenex, integrate potential contacts' context information in an interface that runs on a desktop computer and a mobile device, respectively [16]. Context-Call [21] and Calls.calm[22] provide callers information about the callee's situation and then rely on callers to make reasonable choices regarding the time and mechanisms for communication. The MIT Active Messenger (AM) monitors a user's incoming email messages, prioritizes them based on calendar and other context information, and forwards them to phones, pagers, fax machines, and other communication channels near users [23]. Ubiquitous Message Delivery (UMD) application from Xerox PARC [24] delivered text messages at the soonest acceptable time via the most appropriate terminal near the recipient. However, none of these systems have the notion of call migration or support synchronous communication between different kinds of devices.

Mercury was inspired by the Intelligent Notification System (INS) developed at IBM Watson Research. INS notifies a mobile user when events of interests occur using a convenient device of the user [20]. One component in INS is the Universal Notification Dispatcher (UND), which routes messages to one of several possible communication devices owned by message recipients based on the context of the recipient and the urgency of the message. UND leverages the same infrastructure context service as Mercury, but supports one-way messaging only. In comparison, Mercury also supports two-way communication and addresses many inherent challenges therein.

7. Conclusions

People-to-people interaction is one of the key activities in a person's daily life. Systems that seamlessly integrate communication end-points would go a long way in improving user experience and fostering people collaboration. Mercury enables both asynchronous messaging and synchronous conversation spanning heterogeneous devices. It allows people to receive in-coming communication on a convenient device and, if necessary, switch to a different device during a conversation. It allows a person to be notified of other parties' unified reachability status. It also uses a person's messaging device to alert the person about an in-coming call, providing the feature of soft ring.

Mercury leverages SIP to manage communication sessions. It exploits dynamic user context to drive user preferences, based on which it proactively announces, routes and migrates calls, and to determine users' reachability. At the same time, it gives users opportunities to explicitly control various aspects of a communication. Mercury preserves user privacy as it does not reveal the communication devices a person is connected to or the device she is using for a particular communication. It also allows users to specify policies that determine which other users have access to their unified reachability status. Further, a person is able to prioritize and filter calls based on call attributes and user context. New types of communication devices can be easily integrated into Mercury by deploying a Device Agent for the new devices. The Device Agent consists of a core logic that is reusable across device types and a small amount of device-specific code for handling interaction with devices. This design makes it easy to incorporate new kinds of devices into the system.

For future work, we plan to conduct a user study of the system, to experiment with more device types, and to investigate support for multi-party communication. We would also like to investigate the use of richer context information to provide more information about the reachability and availability of users (e.g. using information about a user's current activity level)

8. References

[1] M. Handley, H. Schulzrinne, E. Schooler and J. Rosenberg. SIP: Session Initiation Protocol. Request for Comments 2543, Internet Engineering Task Force, March 1999.

[2] M. Roussopoulos, P. Maniatis, E. Swierk, K. Lai, G. Appenzeller and M. Baker. Personal-level Routing in the Mobile People Architecture. *Proceedings of the USENIX Symposium on Internet Technologies and System*s, October 1999.

[3] B. Raman, R. Katz, and A. Joseph. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network. *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, CA, December 2000.

[4] H. Lei, D. Sow, J. Davis II, G. Banaduth and M. Ebling. The Design and Applications of a Context Service. *ACM Mobile Computing and Communications Review (MC2R*), 6(4), October 2002.

[5] H. Schulzrinne and E. Wedlund. Application-Layer Mobility using SIP. *ACM Mobile Computing and Communications Review (MC^{2}R)*, Volume 4, Number 3, July 2000.

[6] J. Rosenberg *et al*. SIP Extensions for Presence. Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[7] R. Sparks. SIP Call Control. Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[8] J. Rosenberg *et al*. SIP Extensions for Instant Messaging. Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[9] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computin*g, 5(1): 4–7, 2001.

[10] IBM/Lotus. Lotus Sametime. *http://www.lotus.com/products/lotussametime.nsf-/wdocs/homepag*e.

[11] K. Singh and H. Schulzrinne. Unified Messaging using SIP and RTSP. *IP Telecom Services Workshop*, Atlanta, Georgia, September, 2000.

[12] R. Credle *et al*. IBM DirectTalk for Windows Implementation Guide. *http://www.-redbooks.ibm.com*.

[13] IBM. Websphere Everyplace Access. *http://www-3.ibm.com/software/pervasive-/products/mobile_apps/ws_everyplace_access.shtml*.

[14] IBM. ViaVoice. *http://www-3.ibm.com/software/speech/index.shtml*.

[15] P. O'Doherty. JAIN^{TM} SIP API Specification. *http://jcp.org/en/jsr/detail?id=32*.

[16] John Tang, Nicole Yankelovich, James "Bo" Begole, Max Van Kleek, Francis Li, and Janak Bhalodia. ConNexus to Awarenex: Extending awareness to mobile users, *CHI 2001,* Seattle, WA, April 2001.

[17] Kris Nagel, Cory D. Kidd, Thomas O'Connell, Anind Dey, and Gregory D. Abowd (2001). The Family Intercom: Developing a Context-Aware Audio Communication System. Ubicomp 2001, LNCS 2201, pp. 176-183, 2001.

[18] J. Myllymaki and S. Edlund. Location Aggregation from Multiple Sources. *Proceedings of Third International Conference on Mobile Data Management,* Singapore, January 2002.

[19] H. Wang *et al*. ICEBERG: an Internet-core Network Architecture for Integrated Communications. *IEEE Personal Communications*, 2000.

[20] V. Bazinette, N.H. Cohen, M.R. Ebling, G.D.H. Hunt, H. Lei, A. Purakayastha, G. Stewart, L. Wong, and D.L. Yeh. An Intelligent Notification System. *IBM Research Technical Report*, TR 22089.

[21] A. Schmidt, A. Takaluoma and J. Mäntyjärvi, "Context-Aware Telephony over WAP," Personal Technologies, vol. 4, no. 4, Sept. 2000, pp. 225–29.

[22] E. R. Pedersen, "Calls.calm: Enabling Caller and Callee to Collaborate," Proc. SIGCHI Conf. Human Factors in Comp. Sys., Apr. 1–4, 2001, Seattle, WA, pp. 235–36.

[23] C. Schmandt et al., "Everywhere Messaging," IBM Sys. J. 2000, vol. 39, nos. 3 and 4, pp. 660–77.

[24] M. Spreitzer and M. Theimer, "Providing Location Informatio in a Ubiquitous Computing Environment," Proc. 14t ACM Symp. Op. Sys. Principles, 1993, pp. 270–83; also I Mobile Computing, T. Imielinski and H. Korth, Eds. Dordrecht, The Netherlands: Kluwer, 1996, pp. 397–423.