MatchBase:
A Development Suite for Efficient Context-Aware Communication

Tom Gross, Simone Braun, Susanne Krause
Faculty of Media
Bauhaus-University Weimar
Bauhausstr. 11
99423 Weimar, Germany
<firstname.lastname>(at)medien.uni-weimar.de

Abstract

In distance communication, there is—despite technical progress in information and communication technology—still an asymmetry and tension between the contacter who spontaneously wants to approach the contactee over distance, and the contactee who wants to avoid inappropriate disruption. We have designed and developed MatchBase—a development suite for efficient context-aware communication. Context-aware communication aims to use computer hardware and software to mediate communication between contacters and contactees according to their actual context, where the contacters' context refers to their current situation and their motivation to initiate the communication and where the contactees' context refers to their current situation and availability. In this paper we present the concept and the implementation of the MatchBase development suite.

1 Introduction

Spontaneous communication and coordination among remote parties is a challenge. There are various approaches, but most of them have unwanted side-effects: having pre-arranged meetings is often too inflexible for spontaneous interaction; sending email via computers or short messages via mobile phones in general leave it unclear when to expect a reply; and calling somebody on the phone fails when the person is unavailability or uninterruptible—despite the increased reachability with mobile phones.

Thus in distant communication, despite technical progress in information and communication technology, there is still an asymmetry and tension between the contacter who spontaneously wants to approach the contactee over distance, and the contactee who wants to avoid inappropriate disruption.

Context-aware communication aims to mediate the communication between contacters and contactees according to their actual context, where the contacters' context refers to their current situation and their motivation to initiate the communication, and where the contactees' context refers to their current situation and availability. In order to make computer-mediated communication efficient and context-aware systems need to meet several requirements; they need to (1) analyse the contactors' contexts, (2) analyse the contactees' contexts, (3) match these contexts in order to minimise the contacters' effort and minimise the contactees' interruption, and (4) adapt the mediation according to the inferred match.

In the research fields of computer-supported cooperative work (CSCW) and computer-mediated communication (CMC) several concepts and tools for social interaction in distributed groups have been developed. In particular, instant messaging systems provide presence and availability information in order to facilitate coordination, and ad-hoc conversations among distributed online users [11, 18]. However, only few approaches consider the contactees' contexts, and none consider the contacters' contexts to adapt mediation.

In mobile and ubiquitous computing (UbiComp) context-aware systems capture information about the environment of a user in order to adapt accordingly [4]. For instance, a mobile personal digital assistant can scan its environment for printers, and inform its user about printing opportunities in the vicinity [2]. Yet, these systems mostly focus on the interaction between single users and their personal environment.

We have designed and developed MatchBase—a novel development suite for efficient context-aware communication systems—that goes beyond existing approaches from CSCW, CMC, and UbiComp. The MatchBase suite consists of: MBSens—a sensor component with various software and hardware sensors; MBMatch—an inference engine processing the captured information; and MBAct—an actuator component with various actuators that adapt the behaviour of the user's applications according to the inference results.

In this paper we first give a brief overview of related work. We then present the concept and the implementation of the MatchBase suite. Finally, we draw conclusions.

2 Related Work

Prior work that is related and relevant for our approach of context-aware communication can mainly be found in interruptive communication and can be grouped into work on sensors, heuristic models, and strategies for dealing with interruption.

2.1 Sensors

Systems managing interruptive communication depend on sensors capturing user information to infer on their situation or context (e.g., about their environment, their workplace, their current tasks).

Basically, hardware sensors capture information from the physical world with devices such as cameras, or microphones; and software sensors capture information from the user's computer.

A feasibility study by Hudson et al. [15] investigated the usefulness and reliability of sensors to estimate interruptibility. On the basis of audio and video recordings sensors were simulated for testing the effectiveness of sensor combinations and of statistical models. The work has shown that interruptibility estimation based on sensor data achieves quite accurate results. Fogarty et al. [6] examined the strength of different sensors for the inference of a user's interruptibility. Speech was detected, door and phone states were captured, and motion was captured. Several software sensors captured keyboard and mouse events, and active and non-active windows on the user's computer. It was shown that by the use of sensors predictions of a user's interruptibility can be made, and that the user's work environment and task influence the selection of sensors.

2.2 Heuristic Models

Heuristic models offer further methods to provide information for mediating interruptive communication.

One *simple* but powerful approach is the analysis and detection of rhythms or temporal patterns of user behaviour by observing their daily work behaviour. The analysis of activity and location rhythms is described in [1, 12] and focused on the detection of recurring periods of inactivity and location transition. The presented algorithm detected when a person usually arrived at the workplace, had lunch, or left the office. From these patterns inferences about a user's presence and availability at the workplace were made. This information was then used to automatically set the user status in an instant messaging tool [10].

A more *complex* model of heuristics is the analysis of a person's context. One system applying patterns to predict users' contexts is the Coordinate system [14]. It aimed to improve communication by forecasting users future presence and availability. It supplied enhanced presence information—for instance, for absent users the approximate time of return was posted on a shared calendar, and their incoming messages could be forwarded (e.g., to their mobile phone). The prediction was based on observed data like conversations, meeting states, keyboard and mouse events, and day of week and time of day.

2.3 Strategies for Dealing with Interruption

Depending on the location of the responsibility for the interruption, two strategies can be identified: contacter responsibility, and system responsibility.

The first strategy—that is, *contacter responsibility*—is applied in social translucent systems. They provide the contacter with information gained by different sensors on the contactee's side and let the contacter decide whether a contact initiation is appropriate.

For instance, the MyVine system used this approach [7]. Speech, location, computer activity, and calendar entries were analysed. From these data the system tried to derive the person's presence and availability. The system then provided the communication partners with information on the other's presence and availability, and suggested suitable communication channels. Unfortunately, the results of a test showed that the system often failed in avoiding unwanted interruptions.

A further social translucent system is MyTeam, which provided availability information based on sensor data [17]. Availability information included users' presence as well as their activity. Keyboard and mouse sensors, network connection sensors, and an active badge sensor system were deployed. Sensor data collection worked independently from the client itself—so, even if the client was not running, sensor data could be collected. Different colours representing presence and availability presented the main information. Additionally, different icons reflected a user's activity (e.g., a computer icon indicated keyboard or mouse activity; or a busy icon indicated 'do not disturb'). In a pilot study the users reported that they liked the information about who was at hand, and following from that the decision of the best communication channel. Some difficulties were reported because of forgotten active badges, which resulted in false information about a user's presence.

The second strategy—that is, *system responsibility*—uses sensors to detect users' interruptibility and to automatically enable or disable communication on the contacters' side.

For instance, BusyBody [13] detected users' interruptibility by calculating their cost of interruption with the use of Bayesian networks based on parameters like mouse events, window states, applications, time of day, day of week, speech detection, and calendar information. The cost of interruption was defined as the willingness to pay to avoid unwanted interruptions. From the calculated cost of interruption the system decided whether an interruption is appropriate.

3 Concept

As discussed in the previous section, current computer-mediated communication systems mainly analyse the contactee's situation and leave the decision to the contacter. In MatchBase we aim to improve the overall efficiency of communication by supporting context-aware communication that regards interruption improvement as bipartite issue concerning both contacter and contactee.

In MatchBase efficient communication works as follows: each participant is equipped with various sensors. When contacters initiate communication—say they send an email or instant message—the system requests and calculates information on the contacter, the contactee, and the message to determine the most efficient communication strategy. The matching component is fed with the respective information and calculates a *Degree of Efficiency* (DOE). The DOE determines the further behaviour of the system: if the DOE is high, the message is delivered immediately; and if the DOE is low the message is held back until the system detects an appropriate time for delivery.

This general context-aware communication approach of MatchBase works for any type of technologically mediated communication such as email, instant messaging, phone calls, or short messages. Yet, currently the supported communication channels are limited to email and instant messages.

In the following we will describe the concepts of the contacter, the contactee, and the matching. Please note that in reality the MatchBase suite supports groups of users with changing roles, where users are sometimes contacters and sometimes contactees. The distinction between contacter and contactee is made for a better elaboration and description of the concept.

3.1 Contacter

For the contacter several factors—related to the message and the person—determine the efficiency of communication.

The *messages types* can be question, answer, or information. For email, a sensor can observe if the user uses the 'Reply' or 'New' button, or another sensor can analyse email subjects. For instant messages, a sensor can analyse the text in the body of the message.

The *message urgency* is difficult to determine. A sensor can analyse the text. Another sensor can analyse calendar entries and try to infer the urgency (e.g., message exchange between two participants of an upcoming meeting is likely to be urgent).

The *message complexity* is defined as the contactee's effort to answer the message. To determine the complexity several parameters have to be considered. The message type may be influencing the complexity as well as the matter of the message: a question is inherently more complex than an answer or an informational email, and a programming request can also be very complex.

The *message matter* is the topic. To examine the matter of a message several sensors can be used and combined, such as text analysis of the message subject and/or its content. The matter of the message can be compared with the current task of the contactee. If these two parameters match, a message may be delivered rather quickly, because this message does not cause a context switch on contactee's side.

The *relation to the contactee* is the social connection of the contacter to the contactee. The message matter can provide some insight on the relation. For instance, if the contacter and the contactee are co-workers on a project or in a group, the matter may be a theme regarding this project or group.

3.2 Contactee

For the contactee several factors influence the efficiency of communication, some of which overlap with the factors of the contacter. Obvious and subtle indicators can examine the contactee's interruptibility.

The *obvious* indicators are, for instance, if the contactee is already in a *conversation* (phone and face-to-face) or in a meeting. Sensors for obtaining obvious user information are phone and speech sensors. If users are talking on the phone or have visitors in their office, they are most likely not willing to accept other incoming requests. Conversations can be detected and mapped to a specific user, if the user has a private office.

Depending on the work environment and the personal habits of the user, the *door status* can provide important information. For instance, some people close their office door, if they do not want to be disturbed.

A majority of users is not open for incoming requests, if they are in a *meeting*. Therefore, sensors utilising data from the user's calendar are suitable to verify, if the user is currently in a meeting.

Subtle indicators are the users' workload, and other information that can indirectly be derived from running applications and open documents. The user's *workload* can be inferred from the types of running applications. For instance, programming applications typically require considerable attention. Consequently, integrated development environments (IDEs) can be classified as requiring high attention. On a whole, if the majority of a user's open applications are classified as 'high attention', we assume that the user currently is not open for interruptions.

The active and front-most application and documents also provide information on the *current task*. For instance, if the front-most application is a word processor the current task could be identified as 'producing a document'. In combination with other open documents' names it may be possible to identify the current task. For instance, if the open document is called 'matchbase_concept.doc' and if one of the user's projects

is called 'MatchBase' there is a high probability that the user is writing a project document.

A further aspect to consider is the *relation to the contacter*. For instance, communication from supervisors may have a higher priority for students than from other students; or communication from the manager may have a higher priority than from co-workers. Sensors for contact management applications and address books can provide this information.

3.3 Matching

In order to achieve improvement of the efficiency of communication, the gained information of the communication partners has to be compared and matched. For comparison and matching the type of parameter is important: we distinguish absolute from relative parameters.

Absolute parameters only refer to one communication partner, and their individual preferences. They can be determined independently of the communication partner. For instance, the importance and urgency of a message is related to the contacter; and the interruptibility is related to the contactee. In order to balance these factors we introduce the parameter *Cost of Enforcement* (COE) on the contacter's side, and *Cost of Interruption* (COI) on the contactee's side. We define COE as the willingness to pay to enforce the delivering of a message in Euros. Similarly, COI is defined as the willingness to pay to avoid an interruption in Euros. For each message users are then asked to specify their CEO and COI respectively.

Relative parameters influence each other and can be compared with each other. For instance, the message matter on the contacter's side can be compared with the current task on the contactee's side. Relative parameters include on the contacter's side the message matter, the message complexity, and the relation to the contactee; and on the contactee's side the current task, and the relation to the contacter.

When calculating the *DOE*, absolute and relative parameters need to be treated differently: absolute parameters are left as they are; relative parameters are weighted and influence absolute parameters positively, negatively, or not at all. Examples of positive influence are: if the message is of low complexity; if the contacter's message matter and the contactee's current task are related; or if the participants have a strong social relation.

The system *reacts* based on the calculated DOE. The DOE is scaled into five values: very high, high, medium, low, and very low. So, the actuator delivers messages with a DOE from medium to very high with different kinds of alerts; and holds back messages with a low or very low DOE. Furthermore, fine-grained reactions are offered (e.g., specific sounds and visual alerts aim to attract the contactee's attention for efficient messages).

Besides the decision when and in which way to deliver messages, it has also to be considered, when to deliver *delayed* messages. Czerwinski et al. [3] found that a favourable moment for an interruption is between two tasks—when the actual task is completed, and a new task is not yet started. So, in MatchBase messages on hold are delivered on task shifts. Since users typically switch tasks frequently it is not necessary to define a maximum delay time for messages.

Furthermore, not only a system reaction on the contactee's side is needed, but also on the *contacter's side*. The contacter is informed about what happened with the message sent on contactee's side—that is, if the message reached the contactee or was held back. In the case that the contactee is offline, the matching calculation is not necessary and the system provides the contacter the information that the contactee is unreachable.

4 Implementation

MatchBase suite was implemented on Mac OS X 10.3.8 using Java VM 1.4.2 and AppleScript version 1.9.3. The MBMatch inference engine was implemented in Java; the MBSens sensor components, and the MBAct actuator components were partly implemented in Java and partly in AppleScript.

In the following the overall architecture of MatchBase, as well as details on the MBSens sensor components, the MBMatch inference engine, and the MBAct actuator components are presented.

4.1 Matching Architecture

The MatchBase architecture was developed with the Sens-ation platform [8] and the PRIMI [9] platform. Sens-ation is a service-oriented platform, which provides tools for developers of context-aware, sensor-based infrastructures. The SensBase reference implementation is used. It offers various interfaces such as SOAP, XML-RPC, HTTP, and sockets for storing and accessing sensor values. PRIMI is an open platform that provides software developers with support for the implementation of novel concepts for instant messaging infrastructures. The PRIMIBase reference implementation provides plug-ins for protocols and graphical user interfaces and basic platform services for logging and awareness.

We use SensBase for communication handling and data storage. Also the matching component is integrated into SensBase. PRIMIBase is used as instant messaging infrastructure. The architecture of the MatchBase suite is a client-server-model (cf. Figure 1).

The *client* side consists of two main parts: MBSens, the sensor components collecting data about the user and composed of the sensors, as well as MBAct, the actuator components controlling and manipulating the email and instant messaging applications.

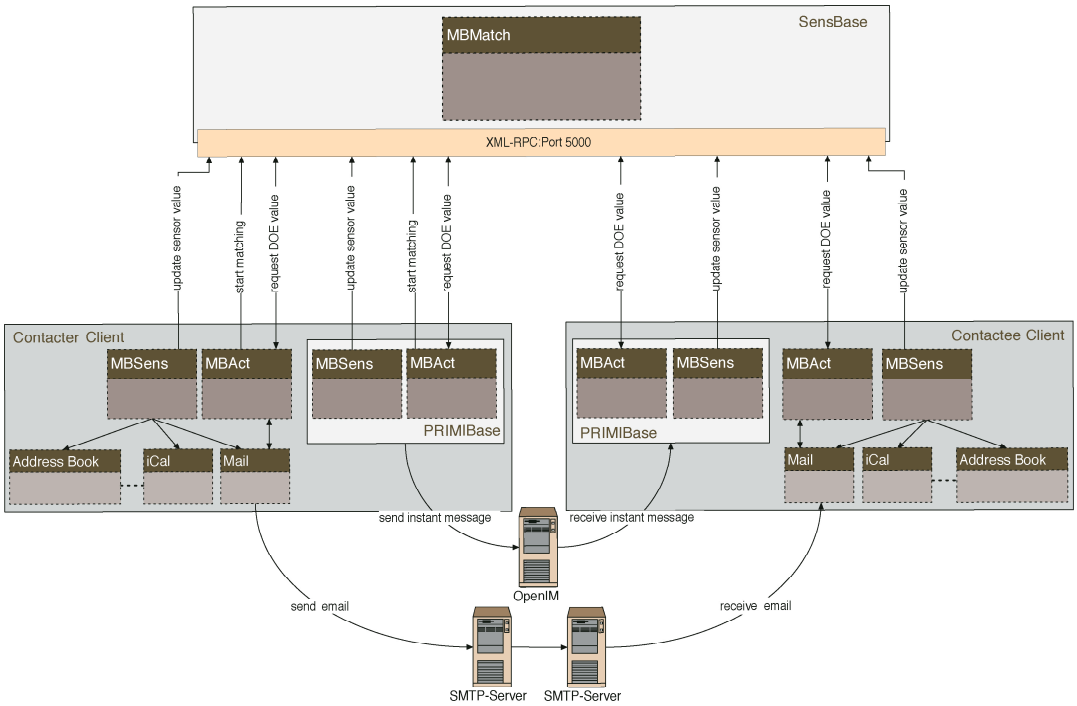

Figure 1. MatchBase system architecture.

The *MBSens* components monitor several applications and are autonomous applications implemented in AppleScript. Additionally, several sensors are integrated in PRIMIBase. Most sensors continuously gather data and send them via XML-RPC calls to SensBase where they are stored. The email and instant message sensors only send data when incoming or outgoing messages occur.

The *MBAct* components control several applications. For instance, they control the Apple Mail application via AppleScript. Additionally, in order to control instant messages, an MBAct Java component is integrated into PRIMIBase.

The SensBase/MBMatch *server* manages all sensors; each sensor is registered with a unique sensor identifier. For easy handling of incoming sensor values, each value is declared as a string data type. In case the sensor provides complex information, the data is formatted as an XML string. This facilitates the subsequent data extraction and preparation for the matching. Java objects are serialised to XML and deserialised from XML with the XStream library [21]. Finally, the server stores all received data in a MySQL database.

When the contacter sends a message, the MBAct components react to outgoing messages by notifying the server via XML-RPC calls, but do not intervene in the regular email or instant message transport. That means that emails are sent to the contacter's SMTP-server, which transmit them to the contactee's SMTP-server, and then in the respective user's email inbox. Similarly, instant messages are delivered to the OpenIM-server, which redirects them to the contactee. On the contactee's side MBAct controls all incoming emails by continuously monitoring the email inbox, and it controls all incoming instant messages by monitoring the traffic. New email or instant messages are held back until their matching results are available. When the contacter sends a message, the MBAct component pushes the server. The server then triggers the MBMatch matching component internally. MBMatch extracts all necessary data from the database and matches the preferences of both communication partners by calculating a DOE for the occurring interruption. The DOE is further provided as sensor. MBAct actuators—of both contacter and contactee—continuously poll this DOE sensor via XML-RPC, and then execute the appropriate action on the held back messages on the contactee's side and inform the contacter about the results of the message delivery (please note that we do not consider the cost of attention of this contacter notification in our calculation). An overview of the workflow of the system on an occurring interruption situation is shown in Figure 2.

In the following the sensor component MBSens, the matching component MBMatch, and the MBAct actuator component are presented in detail.

4.2 MBSens

MatchBase offers hardware sensors capturing information about the users' physical environment, and software sensors capturing information about the users computer and its processes.

Five *hardware sensors* have been implemented: door, phone, movement, and speech sensors.

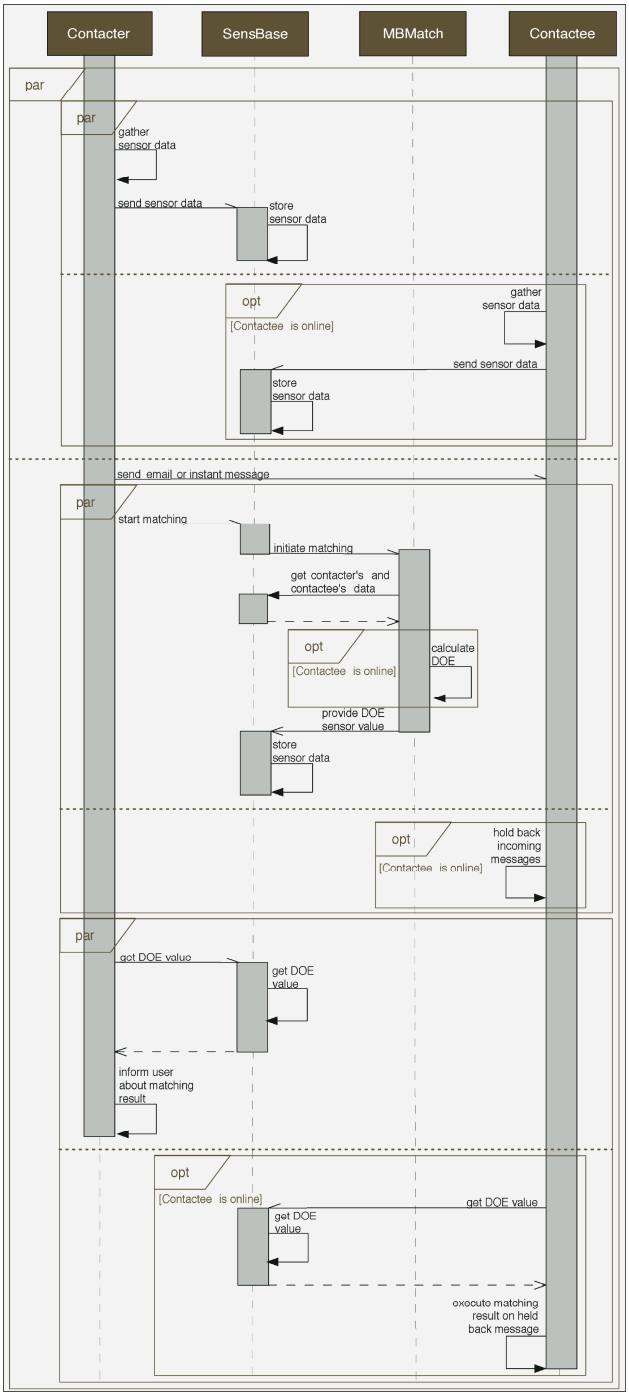

Figure 2. MatchBase workflow.

The *door sensor* gives information about the actual state of a door—that is, if it is closed or opened. The *phone sensor* indicates if the phone receiver is hang up or lifted up. Both are realised with a simple reed switch, one at the door and one at the receiver. Reed switches are sufficient for detecting binary states. The reed switches are connected with an Atmel AVR Microcontroller ATmega8L. It registers the state of the door and the phone respectively (circuit closed or not) and translates them to an 8-bit signal. The signal is sent to the computer via an RS232 serial interface. At the computer MBSens detects and processes the information. The microcontroller continuously sends the states of the door and the phone. A thread listens to the serial port, which is implemented with the RXTX library [16], a native library providing serial and parallel communication for the Java. If the state changes, the actual state of the door and phone are sent to the server via XML-RPC.

The *movement* sensor detects motion in a range of eight meters. It is realised with the Embedded Sensor Board (ESB) [5]. The ESB combines various sensors for motion, temperature, vibration, and luminosity. It offers communication via a serial interface, infrared, and GSM/mobile phones. The ESB is connected via a serial interface to the computer. MBSens handles these data, and submits changes in movement to the server.

The *speech sensor* determines whether users are present and talking. Any microphone can be used—either internal microphones of computers, or external USB microphones. The incoming audio stream is analysed in real-time by an MBSens speech sensor, which was developed in Java based on the Sphinx-4 speech recognition system [19] and which is integrated in PRIMIBase. In order to distinguish conversations from ambient noise, a threshold of 30 seconds is used—that is, only audio signals with a duration of more than 30 seconds are considered as conversations. Furthermore, we defined a five seconds threshold for breaks between words and sentences and a ten second threshold to identify the end of conversations. In our experiments, detection works well with this configuration. For our approach it is sufficient to determine if conversation takes place or not—we do not record audio streams nor recognise words or sentences of conversation.

Several *software sensors* have been implemented: they mostly capture information about applications and the operating system either via AppleScript, or via standard UNIX terminal commands.

An *email sensor* monitors incoming and outgoing emails of the Apple Mail application. It is implemented as a small AppleScript application monitoring the Apple Mail application. When a user sends or receives an email, the names of the sender and recipient are logged. With a simple text analysis of the subject and content the message type and message matter are analysed. Additionally, the title of the email subject and the names of eventually attached files are stored.

The *instant message sensor* is notified when a message arrives or is sent. It is a Java component integrated into PRIMIBase. It generates an XML-string containing information about the contacter, the contactee, the message type, and the message matter; and transfers them to the server.

A *relation sensor* is an AppleScript application that parses all entries of the Apple Address Book application. It extracts the name, email address, and nickname for instant messaging of the owner as well as of any other

existing entries. This sensor sends its data once a day to the server, because frequent chances are not likely to occur or are not vital for our purposes.

The *presence sensor* is a Java component in PRIMIBase that captures the users' login and logout behaviour as well as status changes, and automatically sends changes to the server.

A *meeting and tasks sensor* parses entries in the Apple iCal calendar application. It is implemented as a small AppleScript application analysing calendar data. Events including their title, location, start and end times, and participants as well as tasks including their priority can be extracted. Furthermore, these sensors can analyse if events are full-day meetings; if they are confirmed, cancelled, or only tentative; and if they are repetitive. These data are sent only once a day. Yet, the *in-meeting sensor* examines if at a certain moment a meeting entry can be found in the calendar.

An *applications sensor* retrieves all open applications. It executes the UNIX terminal command `ps` and returns all running processes with their executable name, percentage of CPU, memory usage, accumulated CPU time, state, and the time the process started. Additionally, the System Events application is used to determine the front most (active) and visible application, as well as the open documents of each application. System Events provides the name, path, format and kind, creation time and last modification, owner, and privileges of documents.

The *mouse idle sensor* and *CPU usage sensor* are implemented as UNIX terminal commands. The mouse idle sensor is a binary sensor indicating whether mouse input has been recognised during the last ten seconds. For this purpose, the UNIX terminal command `ioreg –c IOHIDSystem` is continuously executed and the idle time is filtered. If the gained period is greater then ten seconds, it is detected as idle and the server is notified. For the CPU usage sensor the UNIX terminal command `top` is executed every minute. The CPU usage value of the user, the system, and idle are filtered and sent to the server.

4.3 MBMatch

MBMatch is the matching component of the MatchBase suite and is a separate package integrated into the SensBase server.

When a user sends an email or instant message the server is notified by an XML-RPC call triggered by MBAct. With this call further information like sender, recipient, and a unique identifier for message identification are transmitted and the matching is initiated. First, the actual sensor data of the contacter and contactee have to be extracted from the database via an SQL query. If the contactee is online, the matching starts and the preferences of both communication partners are inferred by the use of Weka machine learning environment [20]. The sensor data are prepared, and fed into Weka. Weka calculates the COE of the contacter, the COI of the contactee, and the overall DOE. The obtained values together with further information like the associated unique message identifier are provided to the contacter and contactee as a separate sensor, which can be requested by a XML-RPC call. Lacking current data of contactees indicates that they are offline and consequently no matching can be done. In this case, only the contacter is informed about the absence of the contactee and that the message will arrive when the contactee uses the email or instant messaging application the next time.

4.4 MBAct

Besides the initiation of the matching when an email or instant message is sent, the MBAct components monitor and manipulate the Apple Mail application and the PRIMIBase application.

The MBAct components are responsible for the reaction to the acquired result on the client side. They continuously poll the DOE sensor for new results through an XML-RPC request at the server.

MBAct components control incoming messages. In the case of Apple Mail, MBAct hides newly arrived emails—it marks them as read and moves them to a separate mailbox. In the case of PRIMIBase, MBAct simply holds new instant messages back, and stores them separately. Furthermore, MBAct disables all sounds and visual alerts in both cases.

If the sender of the message is not a participant of MatchBase, the message is delivered right away, because no matching can be done. This indeed means that in some cases contacters who use standard email and instant messaging applications without the MatchBase extension are privileged, since in any case their email is delivered right away. For future scenarios, this effect could be counter-balanced by introducing white-lists for MatchBase users and black-lists for non-users that might mark, delay, or block messages of non-users per default.

Once the matching result for a message is available, the MBAct components execute the appropriate actions. If the message has a very high DOE, the message is displayed—that is, for Apple Mail MBAct moves the email back to the inbox and marks it as unread, it activates sounds and visual signs, and a popup window informs the contactee about the new message. If the DOE is high, the message is also displayed with sounds and visual signs. If the DOE has a medium value, the message is only displayed with a visual sign. In the case of a low and very low DOE, the message is still delayed. These latter messages are not delivered until the actuator components detect the end of the user's current task. For this purpose the running applications are monitored. If one is quit, it is assumed that the user has finished a task and that it is a favourable moment to deliver delayed messages.

And, finally, MBAct notifies the contacter of the delivery of the email message.

5 Conclusions

In this paper we have presented the design and implementation of the MatchBase suite consisting of MBSens sensor components, MBAct actuator components, and the MBMatch inference engine.

On a whole MatchBase provides convenient building blocks for developers of context-aware communication systems. It allows fast and easy development of context-aware communication systems.

In this paper we could not explain the details of the matching and machine learning aspects: the page limit given did not allow for covering that.

Acknowledgments

We thank Tareg Egla and Christoph Oemig, and the Cooperative Media Lab (CML) students for contributing to the concepts and implementation of MatchBase; and the anonymous reviewers for valuable comments.

References

1. Begole, J.B., Tang, J.C., Smith, R.B. and Yankelovich, N. Work Rhythms Analysing Visualisations of Awareness Histories of Distributed Groups. In Proceedings of the ACM 2002 Conference on Computer-Supported Cooperative Work - CSCW 2002 (Nov. 16-20, New Orleans, LO). ACM, 2002. pp. 334-343.
2. Chalmers, M. A Historical View of Context. Computer Supported Cooperative Work: The Journal of Collaborative Computing 13, 3-4 (Aug. 2004). pp. 223-247.
3. Czerwinski, M., Cutrell, E. and Horvitz, E. Instant Messaging: Effects of Relevance and Timing. In Proceedings of the 14th Annual Conference of the British HCI Group - HCI 2000 (Sept. 5-8, Sunderland, UK). ACM, 2000.
4. Dey, A.K. and Mankoff, J. Designing Mediation for Context-Aware Applications. ACM Transactions on Computer-Human Interaction 12, 1 (Mar. 2005). pp. 53-80.
5. ESB. Sensorboards Documentation. Freie Universitaet Berlin, Germany, http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/esb/index.shtml, 2005. (Accessed 19/10/2005).
6. Fogarty, J., Hudson, S.E. and Lai, J. Examining the Robustness of Sensor-Based Statistical Models of Human Interruptibility. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 2004 (Apr. 24-29, Vienna, Austria). ACM, 2004. pp. 207-214.
7. Fogarty, J., Lai, J. and Christensen, J. Presence versus Availability: The Design and Evaluation of a Context-Aware Communication Client. Human-Computer Studies 61, 3 (Sept. 2004). pp. 299-317.
8. Gross, T., Egla, T. and Marquardt, N. Sens-ation: A Service-Oriented Platform for the Development of Sensor-Based Infrastructures. International Journal of Internet Protocol Technology (IJIPT) (accepted).
9. Gross, T. and Oemig, C. PRIMI: An Open Platform for the Rapid and Easy Development of Instant Messaging Infrastructures. In Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications - SEAA 2005 (Aug. 30-Sept. 3, Oporto, Portugal). IEEE Computer Society Press, 2005. pp. 460-467.
10. Gross, T. and Oemig, C. PRIMInality: Towards Human-Centred Instant Messaging Infrastructures. In Mensch & Computer - 5. Fachuebergreifende Konferenz - M&C 2005 (Sept. 4-7, Linz, Austria). Oldenbourg, 2005. pp. 71-80.
11. Gross, T. and Prinz, W. Modelling Shared Contexts in Cooperative Environments: Concept, Implementation, and Evaluation. Computer Supported Cooperative Work: The Journal of Collaborative Computing 13, 3-4 (Aug. 2004). pp. 283-303.
12. Hill, R. and Begole, J.B. Activity Rhythm Detection and Modelling. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2003 (Apr. 5-10, Fort Lauderdale, Florida). ACM, 2003. pp. 782-783.
13. Horvitz, E., Koch, P. and Apacible, J. BusyBody: Creating and Fielding Personalised Models of the Cost of Interruption. In Proceedings of the ACM 2004 Conference on Computer-Supported Cooperative Work - CSCW 2004 (Nov. 6-10, Chicago, IL). ACM, 2004. pp. 507-510.
14. Horvitz, E., Koch, P., Kadie, C.M. and Jacobs, A. Coordinate: Probabilistic Forecasting of Presence and Availability. In Proceedings of the Eighteenth Conference on Uncertainty and Artificial Intelligence (Aug. 1-4, Edmont, Alberta, Canada). Morgan Kaufmann Publishers, 2002. pp. 224-233.
15. Hudson, J.M., Fogarty, J., Atkeson, C.G., Abrahami, D., Forlizzi, J., Kiesler, S., Lee, J. and Yang, J. Predicting Human Interruptibility with Sensors: A Wizard of Oz Feasibility Study. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 2003 (Apr. 5-10, Minneapolis, Minnesota). ACM, 2003. pp. 257-264.
16. Jarvi, T. RXTX: Serial and Parallel I/O Libraries Supporting Sun's CommAPI. http://www.rxtx.org/, 2005. (Accessed 19/10/2005).
17. Lai, J., Yoshihama, S., Bridgman, T., Podlaseek, M., Chou, P. and Wong, D. MyTeam: Availability Awareness Through the Use of Sensor Data. In Proceedings of the Nineth IFIP TC.13 International Conference on Human-Computer Interaction - INTERACT 2003 (Sept. 1-5, Zurich, CH). IOS Press, 2003. pp. 503-510.
18. Nardi, B.A., Whittaker, S. and Bradner, E. Interaction and Outeraction: Instant Messaging in Action. In Proceedings of the Conference on Computer-Supported Cooperative Work - CSCW 2000 (Dec. 2-6, Philadelphia, PE). ACM, 2000. pp. 79-88.
19. Walker, W.F., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P. and Woelfel, J. Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Sun Microsystems, http://research.sun.com/techrep/2004/smli_tr-2004-139.pdf, 2005. (Accessed 19/10/2005).
20. Witten, I.H. and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, 2000.
21. XStream. XStream - About XStream. http://xstream.codehaus.org/, 2005. (Accessed 19/10/2005).