ConChat: A Context-Aware Chat Program

ConChat is a context-aware chat program that enriches electronic communication by providing contextual information and resolving potential semantic conflicts between users.

Anand Ranganathan, Roy H. Campbell, Arathi Ravi, and Anupama Mahajan
University of Illinois at Urbana-Champaign

When two people chat face to face, they know the conversation's context—each knows, for example, the other person's location, what he or she is doing, who else is nearby, and the room's condition (lighting, sound, and so forth). There is also significant communication through facial and other nonverbal cues. Knowing such information makes the conversation richer. However, when you chat with someone electronically, you don't get to exchange this type of contextual information.

Recently, there has been great interest in making applications context aware so that they can adapt to different situations and be more receptive to users' needs. In particular, text-based chat programs could greatly benefit from being context aware. Today's chat programs let users set their status (such as "out to lunch" or "on the phone"), but they generally don't let the two parties exchange any other type of contextual information. Systems such as Babble[1] and Hubbub[2] provide some contextual information (see the "Related Work" sidebar), but they don't let users share the wide variety of contexts that could be sensed in a face-to-face conversation.

With the advent of pervasive computing, we now have sensors that can detect a wide variety of contexts. For example, sensors can detect who is in a room as well as the room's temperature, lighting, and noise level. Recent work even lets us sense the user's mood via emotion-recognition software[3] that interprets facial expressions. If both parties are in pervasive computing environments, they can sense and exchange such contextual information. Our context-aware chat program, ConChat, lets users query their chat partner's context through a side channel—that is, the users can exchange contextual information outside the main channel of conversation.

ConChat uses contextual information to improve electronic communication. Using contextual cues, users can infer during a conversation what the other person is doing and what is happening in his or her immediate surroundings. For example, if a user learns that the other person is talking with somebody else or is involved in some urgent activity, he or she knows to expect a slower response. Conversely, if the user learns that the other person is sitting in a meeting directly related to the conversation, he or she then knows to respond more quickly. Also, by informing users about the other person's context and tagging potentially ambiguous chat messages, ConChat explores how context can improve electronic communication by reducing semantic conflicts.

The context model

Transferring context between parties requires a common model for context. ConChat's model is based on first-order predicate calculus and Boolean algebra. It covers the wide variety of available contexts and supports various operations, such as conjunction and disjunction of contexts and quantifiers on contexts. It allows the creation of complex first-order expressions involving context, so it is possible to write various rules, prove theorems, and evaluate queries.

1536-1268/02/$17.00 © 2002 IEEE

Basic structure: The atomic context

We represent context through a first-order predicate with four arguments. We borrowed the predicate's structure from English, where a simple sentence often takes the form of <subject> <verb> <object>. An atomic context uses this structure along with a context type. We define it as

Context(<ContextType>, <Subject>, <Relater>, <Object>)

Using such atomic contexts, we can construct more complex contexts (discussed later). ContextType refers to the type of context the predicate is describing, Subject is the person, place, or thing with which the context is concerned, Object is a value associated with the subject, and Relater is something that relates the subject and object. Relater can be a comparison operator (such as =, >, or <), verb, or preposition.

Example context predicates include

- Context(Location, Chris, Entering, Room 3231)
- Context(Temperature, Room 3231, Is, 98 F)
- Context(Social Relationship, Venus, Sister, Serena)
- Context(Stock Quote, Msft, >, $60)
- Context(Printer Status, Srgalw1 Printer Queue, Is, Empty)
- Context(Time, New York, Is, 12:00 01/01/01)

The values of the Subject, Relater, and Object arguments depend on the ContextType argument's value. For example, if the ContextType is "location," then Subject can be a person or object, Relater can be a preposition such as "entering," "leaving," or "in," and Object must be a location.

Although this model for context is simple, it can express most basic context types. Furthermore, it is independent of any implementation details such as programming language, operating system, or middleware.

Operations on contexts

The power of a model based on first-order logic is that it is possible to perform complex operations such as Boolean operations and quantifications on context predicates. These operations let us express some of the more complex situations we see in real life.

Boolean operations. We can construct more complex context expressions by performing Boolean operations such as conjunction, disjunction, and negation over context predicates. For example,

Context(Location, Manuel, Entering, Room 3211) $\wedge$ Context(Social Activity, Meeting, In, Room 3211)

refers to the context that Manuel is entering Room 3211, where a meeting is occurring.

Context(Environment Lighting, Room 3234, Is, Off) $\vee$ Context(Environment Lighting, Room 3234, Is, Dim)

explains that Room 3234's lighting is either off or dim.

NOT Context(Location, Manuel, In, Room 3211)

states that Manuel is not in Room 3211.

For ease of representation, we can combine one of the arguments when the other arguments are the same. For example, if the type, relater, and object fields of two different predicates in an expression are the same, we can combine their subject fields, leaving us with just one predicate. For example, we can write

Context(Environment Lighting, Room 3234, Is, Off) $\vee$ Context(Environment Lighting, Room 3234, Is, Dim)

as

Context(Environment Lighting, Room 3234, Is, Off $\vee$ Dim)

Quantification. One or more arguments of the context predicate can be a variable and then quantify over this variable. This lets us parameterize the context and represent a much richer set of contexts. The model allows both universal and existential quantification over variables.

The existential quantifier ("there exists") indicates that the context that follows is true for at least one value of the variable in the variable's indicated scope. Thus, $\exists_S$ X Context(t,x,r,o) is true if and only if Context(t,x,r,o) is true for some value of *x* belonging to the set *S*. For example, to express that Chris is in some location, we can write $\exists_{Location}$ Y Context(Location, Chris, In, Y).

The universal quantifier ("for all") indicates that the context that follows is true for all values of the variable that lie in the variable's scope. Thus, $\forall_S$ X Context(t,x,r,o) is true if and only if Context(t,x,r,o) is true for some value of X belonging to the set *S*. For example, to refer to all people in Room 3231, we write an expression of the form $\forall_{People}$ X Context(Location, X, In, Room 3231).

We can easily construct more complex contexts by performing Boolean operations and quantifications on context predicates. For example, a room controller application could associate the context $\exists_{Person}$ S Context(Location, S, Entering, Room 3234) with the action of playing a welcome message whenever a person enters Room 3234.

The model uses the many-sorted logic model, where quantification is done only over a specific domain of values. That is, we define various sets of values (such as person, location, stock symbol, and so forth). Thus, the Person set consists of the names of all people in our system, Location consists of all valid locations in our system (such as room numbers and hallways), and Stock Symbol consists of all stock symbols in which the system is interested (for example, IBM, MSFT, SUNW, and so on). Each of these sets is finite, and we quantify variables over the values of one of these sets. Because quantification is done only over finite sets, evaluations of expressions with quantifications will always terminate, giving a definite answer.

ConChat

ConChat lets applications obtain a variety of contextual information through its smart spaces infrastructure, Gaia (see http://choices.cs.uiuc.edu/gaia).[4–6] Smart spaces are ubiquitous computing environments that encompass physical spaces. They require a software infrastructure that turns traditional spaces into programmable entities. Named after the Greek goddess of Earth, who embodied the idea of a single, self-sufficient superorganism, Gaia offers services to manage and program a space and its associated state. It also offers services to discover entities (both digital and physical) contained in the space and to store information about those entities

Related Work

Researchers have studied how to gather contexts and use them to make applications more user friendly. The Context Toolkit[1] that Anind Dey developed consists of context widgets that sense context, interpreters that provide higher-level contexts, aggregators that aggregate related contexts, services that execute actions on behalf of applications, and discoverers that find various components. The Context Toolkit, which applications such as the CybreMinder[2] and Conference Assistant[3] use, obtains contexts and then takes action based on the contexts. However, it does not provide a generic mechanism for writing rules about contexts or doing context-based transformations of text.

Paul Castro and his colleagues[4,5] have worked on developing fusion services, which extract and infer useful context information from sensor data using Bayesian networks. ConChat can also use such machine-learning–based approaches (instead of the current rule-based implementation) to determine the user's activities and how busy he or she is. Although machine-learning approaches are more flexible than static rule-based approaches, they require a fair bit of training before they can become effective.

John McCarthy has discussed the problem of transforming data depending on the context.[6] He proposed using lifting axioms to relate the truth in one context to truth in another. Vipul Kashyap and Amit Shet[7] discuss the role of contexts and ontologies for semantic interoperability. According to their view, contexts are used to abstract from the content of an information repository. So-called metadata contexts describe a repository's information context. Researchers have also extensively studied semantic interoperability in the context of the Semantic Web,[8] which aims to enable different parties on the Web to understand the semantics of the material that exists on the Web. Some efforts in this direction include developing languages that describe the semantics of resources such as RDF (see www.w3.org/RDF), development of ontologies,[9] and ways of interoperating between different ontologies.[10] However, all work in this area, so far, has concentrated on achieving semantic interoperability between information systems where there is a fixed schema and a static context. Achieving semantic interoperability between humans is more difficult, because natural language has no fixed schema and the contexts surrounding humans keep changing.

The Affective Computing Projects in MIT's Media Lab try to recognize emotions in individuals and communicate them suitably to other people (http://affect.media.mit.edu). The Babble system[11] transmits social cues such as audience size and how actively people are participating in a multiparty chat scenario. It differs from ConChat in that it presents virtual context to chat participants as opposed to the physical context ConChat presents. The Hubbub[12] system uses sounds to give awareness cues of other people. However, none of these systems provide a generic way of collecting or reasoning about different types of contexts or trying to reduce ambiguities in conversation.

REFERENCES

1. A.K. Dey and G.D. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," *Proc. Workshop Software Eng. for Wearable and Pervasive Computing*, ACM Press, New York, 2000, pp. 434–441.
2. A.K. Dey and G.D. Abowd, "CybreMinder: A Context-Aware System for Supporting Reminders," *Proc. 2nd Int'l Symp. Handheld and Ubiquitous Computing* (HUC 2K), Springer-Verlag, New York, 2000, pp. 172–186.
3. A.K. Dey et al., "The Conference Assistant: Combining Context-Awareness with Wearable Computing," *Proc. 3rd Int'l Symp. Wearable Computers* (ISWC 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 21–28.
4. P. Castro and M. Richard, "Managing Context for Smart Spaces," *IEEE Personal Comm.*, vol. 7, no. 5, Oct. 2000, pp. 21–28.
5. P. Castro et al., "Managing Context for Internet Video Conferences: The Multimedia Internet Recorder and Archive," *Proc. SPIE*, vol. 2969, Jan. 2000.
6. J. McCarthy, "Notes on Formalizing Context," *Proc. 13th Int'l Joint Conf. Artificial Intelligence* (IJCAI 93), vol. 1, Morgan Kaufmann, San Francisco, Calif., 1993, pp. 555–560.
7. V. Kashyap and A. Sheth, "Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies," *Cooperative Information Systems: Current Trends and Applications*, M. Papazoglou and G.Schlageter, eds., Academic Press, London, 1996, pp. 139–178.
8. T. Berners-Lee et al., "A New Form of Web Content that Is Meaningful to Computers Will Unleash a Revolution of New Possibilities," *Scientific Amer.*, May 2001, www.scientificamerican.com/2001/0501issue/0501berners-lee.html.
9. N. Guarino, "Formal Ontology and Information Systems," *Proc. 1st Int'l Conf. Formal Ontology in Information Systems*, N. Guarino, ed., IOS Press, Amsterdam, 1998, pp. 3–15; www.ladseb.pd.cnr.it/infor/Ontology/Papers/FOIS98.pdf.
10. G. Wiederhold, *Intelligent Integration of Information*, Kluwer Academic Press, Boston, 1996.
11. T. Erickson et al., "Socially Translucent Systems: Social Proxies, Persistent Conversation, and the Design of Babble," *Proc. Human Factors in Computing Systems* (CHI 99), ACM Press, New York, 1999, pp. 72–79.
12. E. Isaacs, A. Walendowski, and D. Ranganathan, "Hubbub: A Sound-Enhanced Mobile Instant Messenger that Supports Awareness and Opportunistic Interactions," *Proc. Conf. Computer-Human Interaction* (CHI 02), ACM Press, New York, 2002, pp. 179–186.

and export them to any other interested services and applications. It uses Corba to enable distributed computing and allows events to be sent between components on event channels. Producers and consumers register in the channel to send and receive events, respectively.

Various components in Gaia, called *context providers*, obtain contextual information from either sensors or other data sources. They then let applications query them about the information. Some context

providers also have an event channel where they send context events continuously. Thus, applications can either query a context provider or listen on the event channel to get context information. Gaia also provides a service called the *context engine* where context providers advertise the context they provide. The context engine plays the role of a lookup service and lets applications find appropriate context providers.

Architecture for context-aware chat

Figure 1 shows how ConChat works in Gaia to enable context-aware chat. In a context-aware chat scenario, we assume that the two parties are in pervasive computing environments A and B (PCE-A and PCE-B). In our implementation, PCE-A and PCE-B are rooms, but they could be larger spaces. Both parties run the ConChat application in their pervasive computing environments—let's call these applications ConChat-A and ConChat-B.

The first issue, as in any chat application, is how ConChat-A and ConChat-B discover each other. Many approaches have been used to solve this problem—using machine IP addresses, email addresses, names of users, and so on. Our focus, though, is not the actual process used for discovery. We use a fairly straightforward scheme, with a central server where each ConChat application registers itself and can find other registered users. The registration contains a TCP-IP address for the application, and two ConChat applications communicate with each other by opening a socket and exchanging messages. Although this centralized approach is not the most scalable way to develop a chat application, it is sufficient for our purpose of exploring how chat clients can exchange contextual information.

Sharing contextual information

We now look at how ConChat sends contextual information between users. In our implementation, we can transfer the

1. Party location
2. Number of other people in the room
3. The identities of the other people in the room
4. Room temperature, light, and sound
5. Other applications and devices running in the room
6. User's mood (such as happy, sad, or excited)
7. User's status (such as "on the phone" or "out to lunch")
8. Activity in the room (such as a meeting, lecture, or presentation)

Each party's ConChat contacts the context engine in its pervasive computing environment and gets references to context providers that can provide the contexts in which it is interested. When ConChat starts, it queries the context providers for the current context and then listens on the event channels associated with the context providers for updates. In our implementation, ConChat relies on these context providers to get the first four contexts listed. Currently, we do not have sensors that can automatically sense the user's mood and status, so we rely on the user to provide this information.

To deduce activity in the room, ConChat uses various other cues such as the number of people in the room, what applications are running, and the sound level in the room. ConChat has various rules written in first-order predicate calculus that it evaluates to determine the room's activity. A few of these rules are

- Context(#people, Room 2401, >=, 3) $\wedge$ Context(Application, PowerPoint, Is, Running) => Context(Room Activity, 2401, Is, Presentation)
- Context(#people, Room 2401, >=, 3) $\wedge$ NOT Context(Application, PowerPoint, Is, Running) => Context(RoomActivity, 2401, Is, Meeting)
- Context(#people, Room 2401, =, 1) $\wedge$ Context(Application, Visual Studio, Is, Running) => Context(Room Activity, 2401, Is, Individual Development)
- Context(#people, Room 2401, >, 5) $\wedge$ Context(Sound Level, Room 2401 , Is, High) $\wedge$ $\exists_{\text{Entertainment-Application}}$ Y Context(Application, Y, Is, Running) => Context(RoomActivity, 2401, Is, Party)

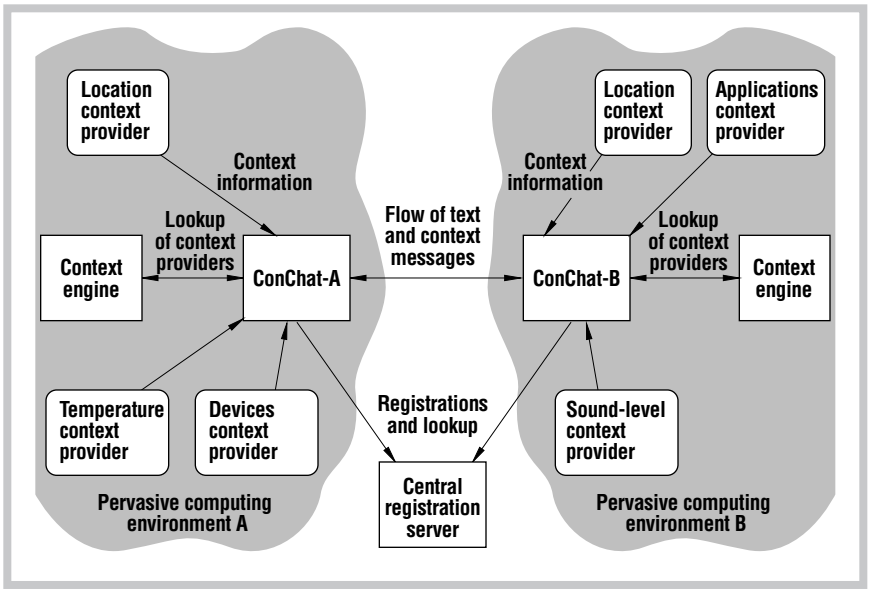
ConChat gets contextual information in the form of a structure containing four fields. This structure represents a context predicate. For example, the location context provider provides information such as Context(Location, Tom, Entering, Room 2401). ConChat maintains a view of its space's current context in the form of a set of true expressions involving context. This information is available to other parties with whom the user might chat.

When user B wants information about some particular context of user A, ConChat-B sends a query to ConChat-A. For example, if user B wants to know who is in the room with A, ConChat-B would send a query of the form Context(Location, X, In, Room 2401) to ConChat-A. ConChat-A has information about who is in room 2401, as provided by the location context provider in its room. It then evaluates the query based on this information, similar to how Prolog evaluates queries—that is, it tries to unify the query with the set of true sentences it has. Once ConChat-A evaluates the query, it sends the results back to ConChat-B, which then displays them to user B.

Alternatively, a user might ask to be notified when a certain condition is satisfied. For example, B could ask to be notified when some other person enters the room. For this, assuming A is in Room 2401, ConChat-B would send an expression of the form $\exists_{People}$ Y Context(Location, Y, Entering, Room 2401) to Con-Chat A. ConChat-A would then notify ConChat-B whenever the expression evaluates to true.

An application like ConChat brings to the fore many privacy issues, because users might not want to expose their context to all parties. However, ConChat lets users specify which contexts can be sent to other users. We use an access control table to determine if a particular user can see a particular context. By default, no information is exposed. A user can change this access control table anytime ConChat is running. For example, when two people who are close to each other (for example, a husband and wife) are chatting, they might not mind exposing all their contextual information. In this case, the husband might get an event when his child enters the room in which his wife is chatting. He might also get information about his wife's mood and the room's noise level and temperature. However, when the wife is chatting with her boss, she might not want to reveal the other

Figure 1. Architecture for context-aware chat.



applications running in her room (so that her boss only knows that she is running the simulation she was supposed to do).

Knowing the other person's context also helps resolve certain ambiguities that might arise during conversation. For example, when one person talks about "Mary" and the other person knows multiple Marys, there could be some confusion. However, based on the context, we might be able to resolve this ambiguity. For example, if a Mary is in the same room as the first person, then it becomes quite clear which Mary the user is referring to. This might not always be the case, but knowing the surrounding context helps improve understanding.

One drawback of conventional chat applications is that users are never quite sure whether the other party is paying full attention to the conversation. Context-based chat can help users decide whether to continue the conversation or chat again later if they find the other party is too involved in other activities. ConChat tries to find out how busy the user is based on what other applications he or she is running and the activities occurring in the room. For example, if the user has multiple ConChat sessions open or is also working on a PowerPoint or Word document, or if a meeting is occurring in the room, the client infers that the user is relatively busy. We're still working on fine-tuning this inference—it's difficult to accurately determine how busy the user really is.

Avoiding semantic conflicts

One problem that plagues all conversations—both face to face and electronic—is ensuring that all parties mean the same thing when they say something. Semantic conflicts typically arise because the two parties are in different contexts. For example, when mentioning a time (say, 8:00 p.m.), the user would typically refer to his or her own time zone. However, if the other party were in a different time zone, that person might interpret the time to be in his or her own time zone. Another example is if an American and a Canadian are chatting, and the Canadian says "$10." The Canadian might have meant 10 Canadian dollars, whereas the American might take it as 10 US dollars.

Researchers have studied semantic interoperability between different systems in the context of integrating heterogeneous databases.[7–9] Many of the approaches in this area involve using enriched schemas that contain semantic information about the entities and their relationships. The problem is different in the case of chat. There is no notion of schema, because natural language is used. Hence, there is far less structure.

There are three main causes of semantic heterogeneity between information systems:[8]

1. *Naming conflicts* occur when naming schemes of information differ significantly (for example, when synonyms are used to refer to the same thing).
2. *Confounding conflicts* occur when information items seem to have the same meaning but differ in reality (for example, a "hot" dish might either mean that the dish is spicy or that its temperature is high).
3. *Scaling conflicts* occur when different reference systems are used to measure a value (for example, price being measured in different currencies).

These causes of ambiguity also exist when humans talk face to face. We try to resolve scaling conflicts by using simple rules to convert between different scales. Naming conflicts are not normally a big problem when humans talk with each other, because they are normally aware of the different synonyms for a word. Confounding conflicts are a bit more difficult to resolve—they require fairly complex natural language processing to disambiguate them. We have managed to resolve simple cases of confounding conflicts such as the words "football" and "hockey," which means different things for Americans and Europeans.

Some sources of semantic ambiguities that occur when humans converse, which are easy to correct given sufficient context information, are

- Time
- Currency
- Units of various measurements
- Date formats (mm/dd/yy versus dd/mm/yy)
- Terms like "football," which mean different things based on context

ConChat has rules for handling each of these semantic ambiguities. It first gathers information about the user's context, using a service in Gaia that provides the location (city and country) of the space, the time zone, and the currency (it could also get the information by contacting Web sites that provide time zone and currency information). It also decides whether the user prefers using the foot-pound-second (FPS) system or the metric system, as well as the format for dates (we use a generic rule where American users prefer the FPS system and the mm/dd/yy format while users in other countries prefer the alternative systems).

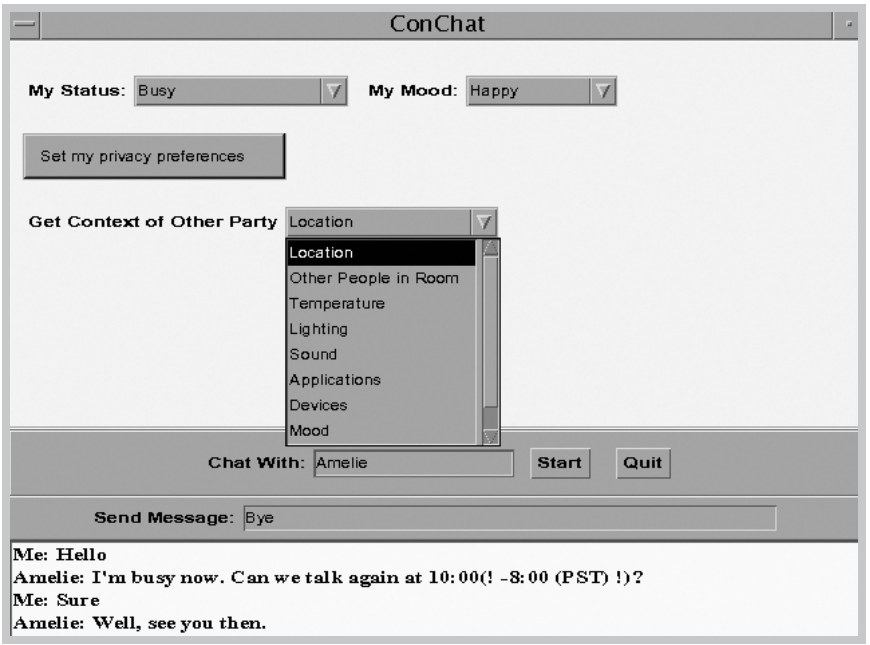


Figure 2. ConChat user interface.

The recipient's ConChat attempts to remove semantic ambiguities in received messages by comparing the sender and recipient's contexts. We use fairly simple pattern-matching algorithms to see if the text contains any ambiguous terms. For example, the presence of a number and currency symbol such as "$" or "Yen" indicates that the conversation is about money. ConChat then tags these potential ambiguities with information about what it thinks the sender meant. In some cases, it converts the text into a format that the receiver will understand correctly; in other cases, it just explains the sender's context. For example, if the phrase "let's talk again at 8:00" is present in the message, it tags the phrase with the sender's time zone and displays the following to the user: "Let's talk again at 8:00 (! –5:00 (EST) !)." This reduces the possibility of misunderstanding. The receiver's ConChat queries the sender's ConChat for his or her time zone. This query occurs just once per session, the first time an ambiguity is detected (it is unlikely that a context such as the user's time zone will change during a session). However, we don't store such contexts across sessions because they might change between sessions.

We use a rule-based system for handling these semantic ambiguities. Some examples include

- $\text{AppearsText}(\text{"Football"}) \wedge \text{Context}(\text{Location-Country}, \text{Sender}, \text{In}, \text{Canada} \vee \text{USA}) \wedge \text{Context}(\text{Location-Country}, \text{Receiver}, \text{In}, \text{France} \vee \text{Germany} \vee \text{UK}) \rightarrow \text{TagWith}(\text{"American Football"})$
- $\text{Context}(\text{Currency}, \text{Sender}, \text{Is}, X) \wedge \text{AppearsMoney}(X, N) \wedge \text{Context}(\text{Currency}, \text{Receiver}, \text{Is}, Y) \wedge (X \neq Y) \rightarrow \text{TagWith}(\text{ConvertCurrency}(N, X, Y))$

The set of rules that ConChat uses to resolve ambiguities are written as a script file. We have developed a simple scripting language modeled on Prolog for expressing these rules. Thus, we can add more rules at any time.

The recipient's ConChat applies these rules whenever the text contains any of the potential ambiguities we have listed. The first rule is applied when "football" appears in the text, the sender is from Canada or the US, and the receiver is in France, Germany, or the UK. The receiver's ConChat gets the sender's country by querying the sender's ConChat.

In the second rule, N, X, and Y are variables (variables are in capital letters). The predicate AppearsMoney evaluates to true if money of value N expressed in currency X appears in the text. The recipient's ConChat gets the sender's currency by querying the sender's ConChat. The convert function converts the number N from the currency of country X to the currency of country Y. It gets the required conversion factor by contacting an appropriate Web site. It would thus tag $10 as $10 (! 11.55 Euro !) if the sender were in the US and the recipient were in France. ConChat evaluates rules involving variables similar to how Prolog handles logic queries—it tries to unify the rule with information it has and, in this process, bind the variables to values (or sets of values).

We have similar rules for converting between the FPS and metric systems. For this, ConChat uses a conversion table, which it stores internally. So, 10 kgs gets tagged as 10 kgs (! 22 pounds !). We also have rules for tagging different date formats. So, 01/10/02 would be tagged as 01/10/02 (! mm/dd/yy !) if the sender were an American and the receiver French.

ConChat is not perfect. Removing all semantic ambiguities from text in English is difficult. ConChat solves only a small part of the problem, but it's a first step toward eliminating semantic conflicts in chat sessions.

User interface

We have developed a user interface for ConChat for experimenting with transferring different types of contexts and resolving various semantic ambiguities. This user interface is not meant for the average end user. Once we have a better understanding of the various issues involved in context-aware chat, we will work on making the interface easier to use and developing better ways of conveying the other party's context.

Figure 2 shows ConChat's user interface. The top panel lets users set their mood and status. Users can also determine the amount of privacy they want by editing an access control table, which determines what contexts are visible and to whom and stores these preferences for future sessions. Users can also choose which contexts of the other party they want to know about. Selecting one of these contexts (such as "Location," or "Other People in Room") gives users the current context. It also lets them specify cer-

tain conditions based on the other party's context and receive a notification when this condition is satisfied. Finally, the bottom panel shows a chat session's transcript. It has an example of tagging an ambiguous time. Only the recipient sees the tags—not the sender. Users can also send instructions to their ConChat client using the Send Message text box. In this way, they can frame their own complex context queries or request certain types of notifications.

There is great potential for future work to produce a chat program that is highly context aware and can make chatting electronically seem almost like chatting face to face. We hope to incorporate different types of sensors into our system—for example, emotion recognition software[3] that can detect the user's mood without the user explicitly entering it.[3] Certain other kinds of contexts include knowing a user's schedule and past activities. We would also like to integrate multimedia capabilities such as audio and video streams and examine if different types of contexts are relevant in such a scenario. Another possibility is multiuser chat.

We also want to improve the user interface and enable better semantic interoperability between the parties. We are exploring how to better convey contextual information and let users easily frame complex context queries on their own. We would like to use natural language processing to determine the context in which something was said (for example, based on what was said before) and indicate this context to the user. We would also like to use other contexts such as the users' history of past activities to disambiguate what they say.

We hope that our experiments with ConChat will lead to a better understanding of how context can be used to improve electronic communication. More specifically, we would like to see which contexts are really useful in enriching communication and how these contexts should be presented to the user. P

ACKNOWLEDGMENTS

Grants NSF CCR 0086094 ITR and NSF 99-72884 EQ from the US National Science Foundation support this research.

REFERENCES

1. T. Erickson et al., "Socially Translucent Systems: Social Proxies, Persistent Conversation, and the Design of Babble," *Proc. Conf. Human Factors in Computing Systems* (CHI 99), ACM Press, New York, 1999.

2. E. Isaacs, A. Walendowski, and D. Ranganathan, "Hubbub: A Sound-Enhanced Mobile Instant Messenger that Supports Awareness and Opportunistic Interactions," *Proc. Conf. Computer-Human Interaction* (CHI 02), ACM Press, New York, 2002, pp. 179–186.

3. I. Cohen, A. Garg, and T.S. Huang, "Emotion Recognition using Multilevel-HMM," *NIPS Workshop Affective Computing*, 2000; www.ifp.uiuc.edu/~ashutosh/papers/NIPS_emotion.pdf.

4. R. Cerqueira et al., "Gaia: A Development Infrastructure for Active Spaces," *Workshop on Application Models and Programming Tools for Ubiquitous Computing*, 2001; http://choices.cs.uiuc.edu/gaia/papers.ubitools01.pdf.

5. M. Román et al., *GaiaOS: An Infrastructure for Active Spaces*, tech. report UIUCDCS-R-2001-2224 UILU-ENG-2001-1731, Dept. of Computer Science, Univ. of Illinois, Urbana-Champaign, 2001.

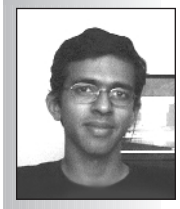
6. M. Román, C.K. Hess, and R.H. Campbell, "Building Applications for Ubiquitous Computing Environments," *Int'l Conf. Pervasive Computing* (Pervasive 2002), Springer-Verlag, Berlin, 2002.

7. H.G. Cheng, *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources*, doctoral dissertation, Sloan School of Management, MIT, Cambridge, Mass., 1997.

8. H.G. Cheng et al., "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information," *TOIS*, vol. 17, no. 3, 1999, pp. 270–293.

9. A. Faquhar et al., "Integrating Information Sources Using Context Logic," *Proc. AAAI-95 Spring Symp. Information Gathering from Distributed Heterogeneous Environments*, AAAI Press, Menlo Park, Calif., 1995.

the AUTHORS



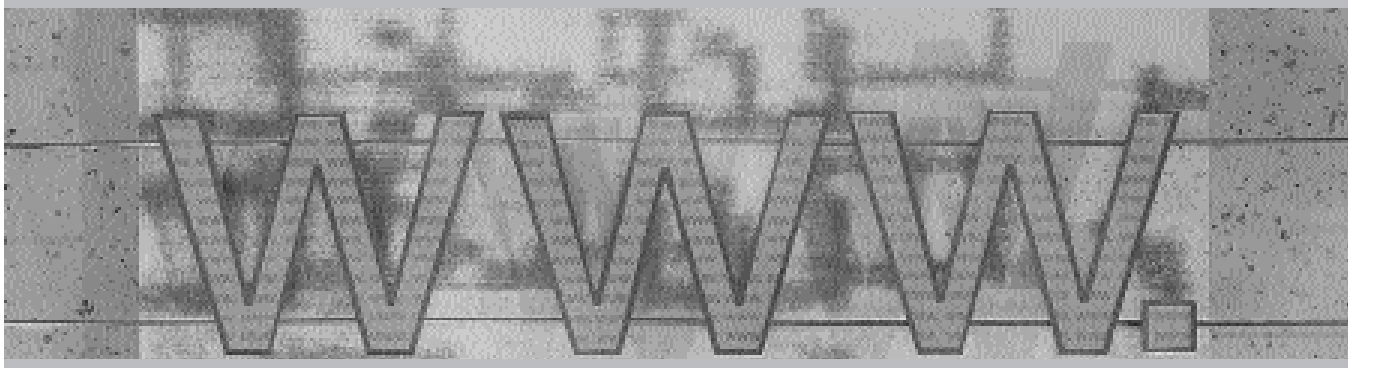
Anand Ranganathan is pursuing his PhD in computer science at the University of Illinois at Urbana-Champaign. His research interests include experimenting with different ways of gathering context and using contextual information to make applications more intelligent in their interaction with humans. He is also interested in the problem of modeling context effectively and trying to infer different types of contexts from sensed data. Other interests include security and data management in pervasive computing environments, mobile computing, and wearable computing. He received his BTech in computer science from the Indian Institute of Technology in Madras, India. Contact him at 3310, DCL, 1304 W. Springfield Ave., Urbana, IL 61801; ranganat@uiuc.edu.

Roy H. Campbell is a professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research interests are the problems, engineering, and construction techniques of complex system software. His current research projects include the integration of video with the Web and its applications, high-performance networks, digital library system software, dynamically adaptable operating systems that support continuous media, mobile computer security, and performance issues of distributed shared-memory multiprocessors. He received his PhD from the University of Newcastle upon Tyne. Contact him at 3125 DCL, 1304 W. Springfield Ave., Urbana, IL 61801; rhc@uiuc.edu.

Arathi Ravi is a graduate student in the Department of Computer Science at the University of Illinois at Urbana-Champaign. Her research interests are in the area of pervasive computing and human-computer interfaces. She is interested in making use of context for developing better interfaces to different applications. She received her BEng in computer science and engineering from Madras University, India. Contact her at 503 E. Stoughton, Apt. #8, Champaign, IL 61820; aravi@uiuc.edu.

Anupama Mahajan is earning her MS in computer science from the University of Illinois at Urbana-Champaign. Her research interests include distributed computing, system and functional verification, and ubiquitous computing. She received the RajaRaman Award for academic excellence in computer science in 2000. She received her BS in computer science from the Thapar Institute of Engineering and Technology in India. Contact her at 302 S. Fourth St., Apt. #12, Champaign, IL 61820; amahajan@uiuc.edu.

To receive regular updates, email
dsonline@computer.org



Relaunched with a New Design

IEEE Distributed Systems Online **supplements the coverage in *IEEE Internet Computing* and *IEEE Pervasive Computing.***

Each monthly issue includes magazine content and issue addenda such as interviews and tutorial examples.

IEEE Distributed Systems Online brings you peer-reviewed features, tutorials, and expert-moderated pages covering a growing spectrum of important topics, including

Grid Computing
❍
Security
❍
Distributed Agents
❍
Middleware
❍
Mobile and Wireless
❍
and more!

IEEE Distributed Systems Online recently relaunched with a new design, and continues to provide news, an events database, book reviews, and more.

Check out

dsonline.computer.org

to keep up with all that's happening in distributed systems.

IEEE distributed systems ONLINE
Expert-authored articles and resources