Negotiating Context Information in Context-Aware Systems

Mohamed Khedr and Ahmed Karmouch, *University of Ottawa*

Context-aware environments must allow adaptive and autonomous access to context information. This multiagent middleware uses a negotiation protocol and ontology model to make the environment more easily personalized at runtime and adapted and managed at provisioning time.

In a pervasive environment, a wide range of devices and resources use heterogeneous networks to perform the tasks involved in spontaneous ad hoc communication. The environment's infrastructure must therefore make available a rich set of computing capabilities and services at all times and in all locations in a transparent, integrated, and convenient way.[1] Context provides perceptual information about the location and status of the people, places, and other devices in the environment.[2]

The design and development of context-aware applications introduce requirements and challenges not found in traditional applications. First, the dynamic, adaptive provisioning of context information requires negotiating context features (location, identity, and activity), engagement (conditions, events, and actions), and dependency. Next, agents and context-aware applications must be able to effectively and reliably use context information during negotiation. So, context-aware systems require these components:

- *Context ontology and inference* to facilitate the sharing, management, and inference of a given context
- *Context-level negotiation* to agree on levels of context that can be provided and enable personal context provisioning
- *Context management* to store, retrieve, query, and modify context information

These characteristics elevate context from an ambiguous concept to a value-added service that tailors levels of context information representation and quality to user needs.

We've developed the *context-level negotiation protocol* to facilitate development of context-aware, personalized applications and an ontology model for representing context and negotiation information. Our context-based conferencing environment demonstrates our approach's effectiveness.

Context and context-level agreement

Pervasive-computing research shows that a user's environment is dynamic, adaptive, and interactive. It contains sensors, wireless devices, and personal and service agents operating autonomously with the aid of context. By context, we mean information about physical characteristics (such as location and network elements), the system (such as applications running and available services), and the user (such as privacy and presence). The environment becomes context-aware when it can capture, interpret, and reason about this information. The vast quantity and diverse quality of context, in addition to the fact that no user is interested in all available context, led us to the concept of *context-level agreement*, in which entities negotiate context specifications using an ontology-based model. CLA's advantages include

- A declarative method with which users state their context requirements and context providers offer dynamic customized context-based service information
- Protection against unexpected performance degradation because context providers know users' needs and can thus better manage their resources
- Seamless context-based services across domains in a large environment
- An easy mechanism with which context providers can intelligently supply and monitor context resources

We define three broad types of CLA:

- *Passive context*. Each user's context specifications are simple, requiring little or no complex operation from the context provider.
- *Active context*. Users' context specifications result from complex content provider operations such as filtering, merging, composing, and monitoring. Negotiation at this level aims to achieve common understanding of the context's meaning and format.
- *Spontaneous context*. Context management and inference must be able to cope with the uncertainty and vagueness inherent at higher context levels. Negotiation at this level lets a personalized group of context providers manage user context requirements such as information persistence, delivery methods, remote access, and inference on each user's context.

For CLA to be dynamic, automated, and extensible, it must provide a framework for negotiating context specifications and an ontology that provides standard and extensible constructs of context and negotiation performatives.

Context-aware system architecture

Figure 1 shows our distributed multiagent system architecture. Agents are lightweight and expected to operate in pervasive environments in which resources and computing nodes have limited capabilities. We implement system agents (such as the context-management agent and the inference agent) in hosts with moderate computing power. Other agents (such as user and service agents) reside in user nodes that can support any available lightweight-agent platform, such as the Lightweight Extensible Agent Platform (LEAP), which is based on the Java Agent Development Framework (JADE, http://jade.tilab.com). In addition, the inference agent and knowledge base repositories span several nodes to decrease bottlenecks and increase scalability and robustness against failure. Agents perform negotiation, query, and service invocation through agent message passing.

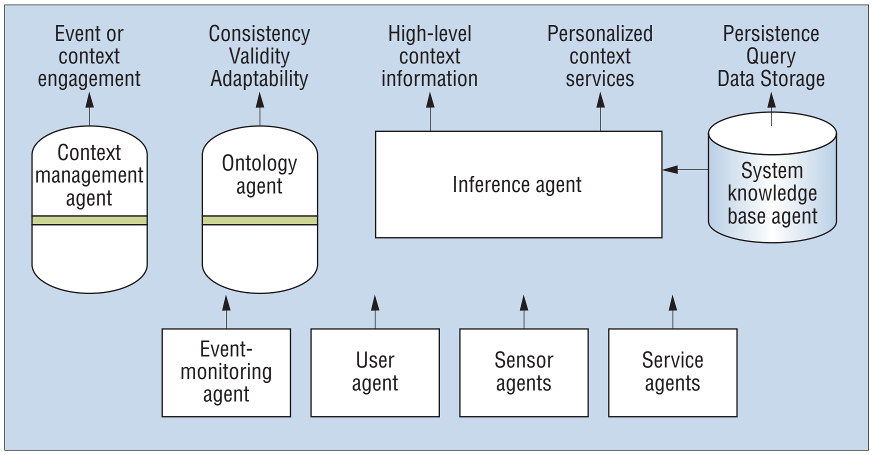
Event or context engagement
Consistency Validity Adaptability
High-level context information
Personalized context services
Persistence Query Data Storage
Context management agent
Ontology agent
Inference agent
System knowledge base agent
Event-monitoring agent
User agent
Sensor agents
Service agents

Figure 1. A multiagent system architecture.

Agent types

Five main agent types perform the functions required for context negotiation and provisioning. We group these functions into three modules, as Figure 2 shows. The *context representation* module gathers context from the surrounding area and provides it in an ontology-based representation that facilitates interpretation and inference. The *context communication* module governs communication between agents using a registration process as well as the negotiation protocol. The *context management* module provides methods for querying and storing context information.

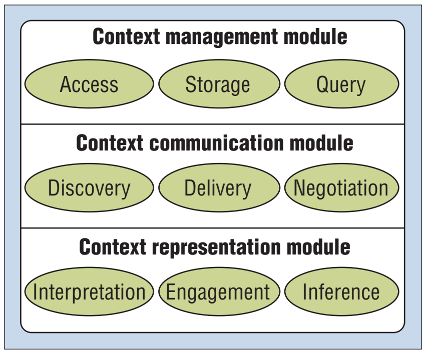
Context management module
Access
Storage
Query
Context communication module
Discovery
Delivery
Negotiation
Context representation module
Interpretation
Engagement
Inference

Figure 2. The multiagent system supports nine functions divided into three modules.

User and service agents. User agents represent users in the environment, negotiating on their behalf to set both the required context specifications and the methods of context representation and delivery. They maintain user profiles consisting of user activities, scheduling, and associated devices, as well as context parameters such as identity, location representation, and personal context information.

Service agents represent services in the environment. They maintain service profiles, which define service capabilities and the knowledge required to negotiate each service's optimal performance according to context.

Context management agent. The context management agent is the system administrator. The CMA negotiates with user agents to achieve CLA and with context provider agents (which we describe later) to schedule and guarantee the required context. After negotiating the context, the CMA monitors ongoing context-based sessions and manages environmental resources. The CMA can also cancel, modify, or renegotiate context specifications because of environmental changes or because agents violate their CLA-negotiated tasks.

Registering with the CMA lets agents project their presence to other environmental entities and obtain the authorization needed to use and negotiate context information. The CMA stores relevant information in a knowledge base repository for inference, consistency, and knowledge sharing. The multiagent system uses this repository to track entities' information and store temporal session information.

Inference agent. The inference agent integrates reasoning, conflict resolution, and coordination. Algorithms from AI, semantic knowledge, and fuzzy logic make the inference process powerful, extensible, and adaptable. The inference agent manages the engagement and inference processes shown in Figure 2. It uses context captured from sensors and user and service profiles as facts in the context inference process, using these facts to build a system knowledge base repository for deducing new context information. The inference agent uses logic-reasoning mechanisms to ensure that captured context instances are consistent with each other and with arguments defined in the context ontology. This also lets us construct an inferred hierarchy of context information based on the designed ontology classes and resolve inconsistencies that can arise from firing rules and triggering actions.

Ontology agent. The ontology agent provides the semantic functionalities that other

Related Work

Few approaches in the context awareness field explicitly focus on context modeling in widely deployed pervasive-computing systems. Rather, most current research concentrates on frameworks and toolkits to support ad hoc context representation, and thus limits the interoperability and ease of integration necessary for providing smart pervasive environments.[1]

The Gaia metaoperating system recognizes the need for context information in a context service component.[2] Its context service lets applications query and register for particular context information, which helps them adapt to their environment. Applications can search a registry of available context providers for providers of their desired context. This approach uses a first-order logic to model context, allowing the system to write rules describing context information. However, this approach doesn't support negotiation between applications and context providers, which could provide flexible context provisioning. Applications can use only the context providers in the registry. Additional application instances will introduce scalability issues such as temporary failure or inability to provide context information because of access overloading.

Wai Yip Lum and Francis Lau propose a context-aware decision engine for content negotiation.[3] They use a proxy server for content modification based on certain context decisions. However, they limit the context definition to quality of service, a significantly more narrow definition than ours, which considers context to be any information affecting an entity (such as location and presence). Because the context definition is limited, the adapted content might still lack context awareness.

Reconfigurable Context-Sensitive Middleware provides the adaptive object container (ADC) for runtime context acquisition.[4] The RCSM interface definition language (IDL) compiler tailors ADCs to particular context-sensitive objects. The context-sensitive interface lists the contexts the applications use, the actions they provide, and a mapping between the contexts and the actions that clearly indicates the values at which an action should be completed.

Our multiagent system outperforms RCSM in two situations. First, our ability to negotiate context using the multiattribute utility function lets us use a just-in-time range of context values when an action is to be performed or completed. The negotiation gives us the flexibility and extensibility to add new context information and modify existing information during runtime. Second, the RCSM has no clear method for composing context semantically or for reasoning with the composed context information.

Other researchers describe a framework that uses *composite capabilities/preferences profiles* to enable communication and negotiation of device capabilities and user preferences in the context of a Web session.[5,6] They use the framework to assert metadata information about Web elements. This approach depends on a negotiation protocol in which intermediate proxies can adapt content to the device profile in advance. In a pervasive environment, nothing can be statically configured. A negotiation mechanism such as ours dynamically sets context information at runtime. Moreover, CC/PP is based on the resource definition framework, whose small set of constructs for representing metadata limits the information that can be presented and inferred. In contrast, our proposed model presents context ontologies in much richer semantic languages, such as DAML+OIL and OWL.

Cobra also uses OWL to define ontologies for pervasive environments.[7] However, our design objective is a straightforward generic ontology for context representation that developers can apply while using the negotiation mechanism to set and modify context specifications. We also provide a powerful inference engine based not only on logic reasoning (as in Cobra) but also on fuzzy and inductive reasoning. We chose this over a Bayesian approach[8] because fuzzy reasoning provides a natural extension to our context ontology through its use of linguistic terms and linguistic hedges.

References

1. N. Davies and H.-W. Gellersen, "Beyond Prototypes: Challenges in Deploying Ubiquitous Systems," *IEEE Pervasive Computing*, vol. 1, no. 1, 2002, pp. 26–35.
2. M. Román et al, "A Middleware Infrastructure for Active Spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, 2002, pp. 74–83.
3. W.Y. Lum and F.C.M. Lau, "A Context-Aware Decision Engine for Content Adaptation," *IEEE Pervasive Computing*, vol. 1, no. 3, 2002, pp. 41–49.
4. S. Yau et al., "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 3, 2002, pp. 33–40.
5. L. Suryanarayana et al., "CC/PP for Content Negotiation and Contextualization," *Proc. 2nd ACM Int'l Conf. Mobile Data Management*, ACM Press, 2001, pp. 239–245.
6. T. Lemlouma et al., "The Negotiation of Multimedia Content Services in Heterogeneous Environments," *Proc. 8th Int'l Conf. Multimedia Modeling,* MMM Press, 2001, pp. 187–206.
7. H. Chen et al., "An Ontology for Context-Aware Pervasive Computing Environments," *Proc. IJCAI 03 Workshop Ontologies and Distributed Systems*, IJCAI Press, 2003.
8. P. Korpipää et al., "Managing Context Information in Mobile Devices," *IEEE Pervasive Computing*, vol. 2, no. 3, 2003, pp. 42–51.

agents can use to represent and share context in the system. At negotiation time, it combines context specifications from agent requests and profiles according to the designed ontology's constructs, axioms, and rules of composition, which it then provides to the CMA. It also provides searching and browsing interfaces so that registered agents can look up and use available ontologies.

Context provider agents. The system wraps each context source with a CPA responsible for capturing the raw data from the source and interpreting it so that it's understandable by other agents using the lookup service that the ontology agent provides. The CPA negotiates context specifications that are captured and acquired by the context source under its control.

The context ontology model

Context-aware applications require a unified context model that is flexible, extensible, and declarative to accommodate a wide variety of context features and dependency relations. Research in context-aware applications[3] (see also the "Related Work" sidebar) defines models based on a top-down architecture in which context is modeled according to the application-layer objectives and functional intentions. These models are ad hoc and have limited extensibility and interoperability with other context-based systems. Ad hoc modeling also limits the use of the multiagent system across domains because not all agents can understand and use context information. Our context-modeling approach is therefore ontology based and represents the generic context types needed in pervasive computing. The ontology is motivated by the need to share knowledge about actors, locations, and schedules so that context-aware applications can trigger actions and infer outcomes.

In our ontology, the root class ContextView provides an organizational point of reference for declaring context information. At least one instance of ContextView exists for each distinct entity in the environment. Its properties are contains and invokes, with the classes ContextFeature and ContextEngagement as the properties' respective ranges. Each ContextView instance contains at least one subclass of ContextFeature, associated with one or more ContextDependency classes and invoked by ContextEngagement.

ContextFeature consists of classes related to the physical environment (such as location), the system (such as services), individuals (such as users), and the underlying network (such as network characteristics).

The Location class interprets an entity's location information as detected by embedded location sensors. The ontology associates location with such concepts as university, company, and generic place. The location class also includes meta-information fields such as location-updating rate, services in the current location, and types of network the location supports.

The Actor class provides information interpreted by identity context providers such as the radio frequency tag system. It has two main subclasses:

- Person has person-related properties detected by the sensors.
- Agent represents the system's software agents.

Information provided includes an ID value, the entity type detected (such as infrared model or RF tag), the entity's role (such as printer agent or student), detection time, and reference to the actor's personal profile for further context acquisition.

The Network class identifies resources, protocols, and topology configurations. We limit our scope to wireless networks, especially 802.11 and Bluetooth. A network profile class containing the resource's capabilities and preferences represents each network resource, such as a gateway server or an access point.

The Service class identifies services discovered in the environment. It includes the service profile (such as a printer profile), service parameters, applications engaging the service, and the service's complexity type.

The Action class represents activities in a user's current location that involve other users or other services detected and projected to the user's ContextView. Activities currently defined in the ontology relate to our proposed implementation scenario and include audio- and videoconferencing, local or remote presentation-based meetings, and regular activities such as emailing, chatting, and file sharing.

Figure 3 shows a partial representation of the ontology, with classes represented by bullets and properties given between brackets. The complete ontology is available at www.daml.org/ontologies/397 and at www.schemaweb.info/schema/SchemaDetails.aspx?id=181.

- Context Feature ()
  - ❍ ActionProfile (actionprofileName, actionStartTime, actionEndTime, actionExpectedDuration, hasInvolvingAgents, hasInvolvingPersons, hasInvolvingServices)
  - ❍ Network (hasNetworkProfile, nDomain, nName)
    - ■ NetworkResource (hasServiceProfile)
      - Laptop ()
      - PDA ()
      - Phone ()
      - Printer ()
  - ❍ NetworkTopology ()
    - ■ Wireless (hasAccessPoint, hasAccessPointAddress)
      - 802.11 (modeOfOperation)
      - Bluetooth (modeOfOperation)
  - ❍ NetworkProfile ()
    - ■ NetworkProtocol (hasProtocolChar, protocolName)
    - ■ NetworkParameter (nParameterName, nParameterValue)
    - ■ ProtocolCharacteristic (pValue)
  - ❍ Physical ()
    - ■ Actors (actorID, actorLocation, actorName, actorRole, hasCurrentAction, usesCurrentService)
      - Agent (hasCommunicationLanguage, hasPlatformType)
      - Person (hasPersonalProfile)
    - ■ Location (altitude, degree, direction, hasCurrentAgents, hasCurrentPersons, hasLoadedServices, hasLocationRange, hasNetworkSupport, hasRestrictedActions, latitude, longitude, placeName)
  - ❍ ServiceCategory ()
    - ■ Application (aName, effect, input, output, precondition)
    - ■ Resource (effect, input, output, precondition, rName)
    - ■ ServiceParameter (sParameterCondition, sParameterName, sParameterValue)
      - ServiceProfile (hasNetworkSupport, hasServiceCategory, hasServiceLocation, hasServiceParameter, hasServiceProfileName, hasServiceActor)
  - ❍ Social ()
    - ■ Action (actionLocation, actionName, hasActionProfile, hasCurrentAgents, hasCurrentPersons, hasNetworkSupport)
      - Conference ()
      - AudioConference (usesCurrentService)
      - VideoConference (usesCurrentService)
      - Meeting (usesCurrentService)
  - ❍ Role ()
    - ■ AgentRole (agentName, agentRole, hasRestrictedActions)
    - ■ PersonRole (hasRestrictedActions, personName, personRole)

Figure 3. A partial representation of the context ontology showing the ontology classes and their properties.

Context-level negotiation protocol

CLNP is a multiattribute-based negotiation protocol that allows automated context identification and complex decision making according to user needs—functions traditionally performed manually. User or service

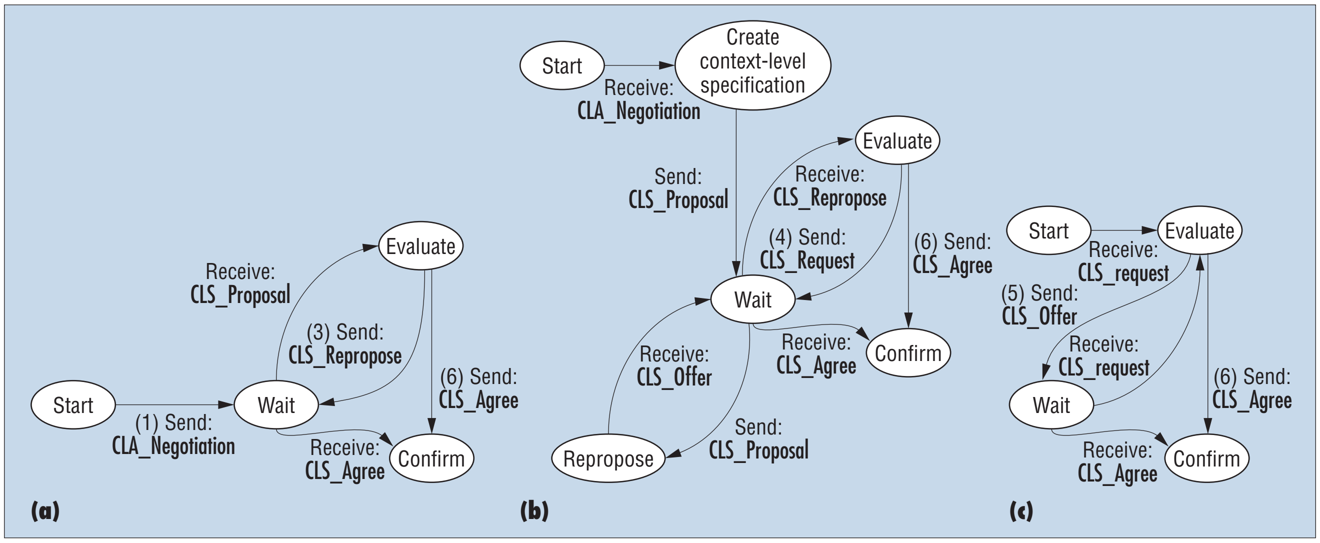

Figure 4. The context-level negotiation protocol from three perspectives: (a) the user agent; (b) the context management agent; (c) the context provider agent. Processes in each part follow and precede processes in the other parts.

agents send CLA requests to the CMA, which maps the CLA to context specification parameters and issues a negotiation request to available CPAs. The CPAs evaluate the proposed specifications and either accept or repropose. Negotiation continues until the agents reach an agreement based on the utility function calculation (described in the next section).

CLNP aims to

- Set the context specifications between user and service agents and CPAs explicitly and declaratively
- Authorize the CMA to register and reserve resources and services implicitly and decide how the CPA will deliver and monitor context
- Set alternatives for unexpected situations implicitly
- Let CPAs schedule context-based tasks implicitly and minimize overloading

Negotiation uses our ontology model. It involves context exchange and specification, a utility function for decision-making, and a procedure for message flow between agents to set context specification values.

The utility function

Quantifying the negotiation process requires normalizing each context specification's range. We then use a multiattribute utility function to model each negotiated context specification. The utility function, which we adopted from the work of Antony Richards and his colleagues,[4] has these characteristics:

- A lower bound on x (the normalized negotiated context parameter), below which user satisfaction is negligible and approximated to zero
- An upper bound on x, above which any gain in user satisfaction is negligible and approximated to one
- Between these two bounds, a satisfaction factor that's a monotonically increasing function of x
- A sensitivity parameter p for projecting user satisfaction

The logarithmic function matches these requirements. We can therefore represent the utility function as a logarithmic function in the form of

$$U(x) = a.\ln(b.x + c) \qquad (1)$$

$$a = \frac{1}{p - A_o} \qquad (2)$$

$$b = \frac{e^{1/a} - 1}{R - A} \qquad (3)$$

$$c = \frac{R - Ae^{1/a}}{R - A}, \qquad (4)$$

where $U(x)$ is the utility function, x is the context parameter values in negotiation, p is the user sensitivity parameter, A_o is the expected context sensitivity, A is the minimum context parameter value, and R is the maximum context parameter value.

The utility function depends on the sensitivity parameter p. For every value of p, agents project their interest in a context specification according to the utility function shape. For example, for p values smaller than A_o, the user is more sensitive to large context parameter values than small values, such as location information in outdoor applications. For p values larger than A_o, the user is more sensitive to small context values than to large values, such as delay in emergency notification.

The CMA uses the negotiation process (discussed in the next section) to determine the working utility curve. To find the optimal utility curve, the CMA applies the following decision equation on the initial context parameter value A_L requested by the user or service agent:

$$U(A_L)_{x=A_L} = a.\ln(b.A_L + c) \qquad (5)$$

If $U(A_L) << U(R)$ for $A_L < R$ and $A_L > [A + R/2]$, then $p < A_o$, the agent is more sensitive to large context values, and the CMA uses p from Equation 2 to satisfy Equation 5.

If $U(A_L) >> U(A)$ for $A_L > A$ and $A_L < [A + R/2]$, then $p > A_o$, the agent is more sensitive to small context values, and the CMA chooses p from Equation 2 to satisfy Equation 5.

After deciding the utility function curve for each context parameter, the CMA assigns a weighted priority value for each context parameter indicating its importance (as perceived by the user) in the overall negotiation. The CPAs and CMA use the priority values to differentiate the levels of context representation and delivery to the user.

The weighted priority value takes the form

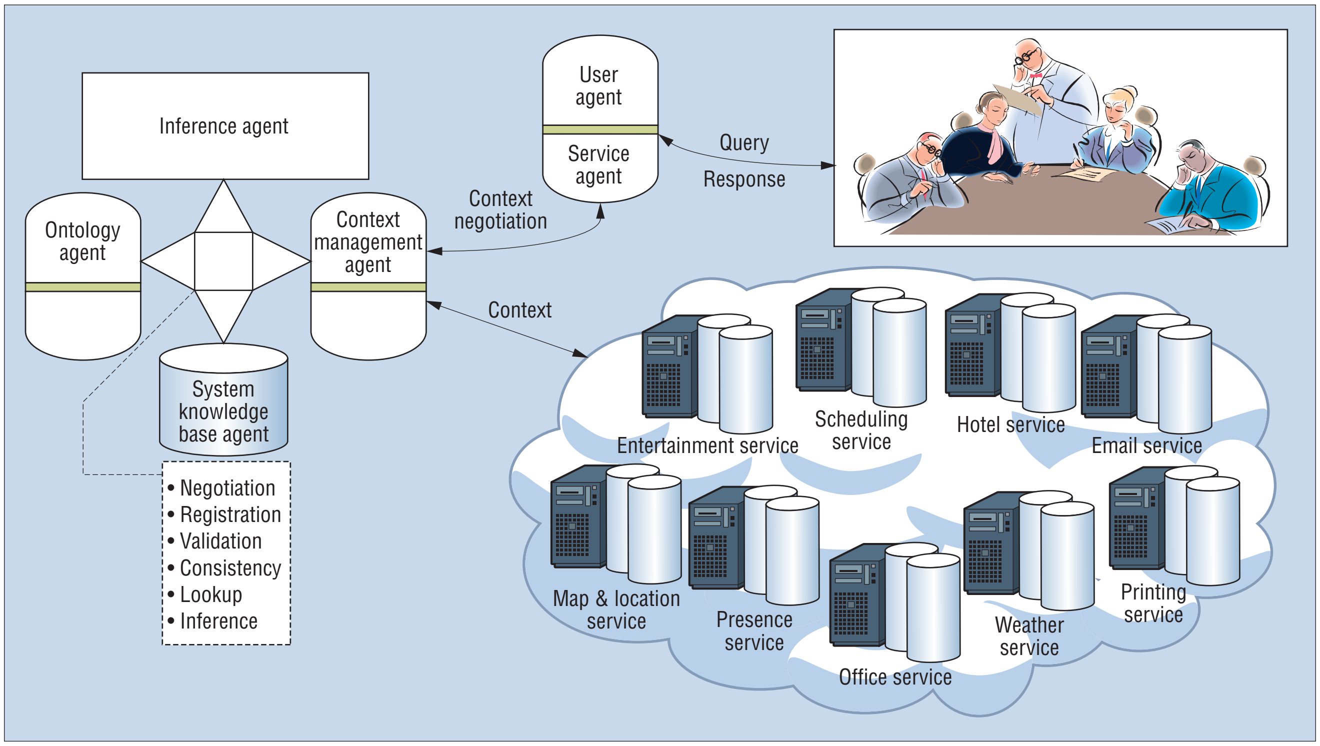

Figure 5. The conference environment integrated with the multiagent system. Available services are either *services in advance* or *services on the fly*, depending on the input context source.

$$W_i = \frac{A_{Li} - A}{R - A},$$

where $i = 1 \ldots N$ and N is the total number of context parameters. The total satisfaction value will equal

$$\sum_i U(x_i).W_i .$$

The CMA uses this value to decide whether the agents have reached agreement or the negotiation should continue.

The negotiation process and message flow

The user or service agent starts the negotiation by sending the CMA a CLA_Negotiation message with the initial context specification ((1) in Figure 4a)—that is, the context specifications defined according to the ontology model. The user or service agent sets specifications, such as location and availability information, and adds an expected deadline for the negotiation.

When the CMA receives the CLA_Negotiation message, it builds a template from the context specifications fed to it by the CPAs. The CMA assigns the utility function parameters $\{a, b, c, A_o, R, A\}$ that match user context specifications using Equations 1 through 4 and the decision functions in Equation 5. After determining the best utility function to use, the CMA builds a composite context template reflecting the user or service agent ContextView. The CMA creates the composite context at runtime using a relation-based binding process containing eight main relationships:

- Consists_of relates the environment with the places under the CMA's management.
- Occupied_by relates the environment with the users currently detected by the sensors.
- Contains associates the environment with the discovered services.
- Composed_of defines subservices composing services.
- Loaded_with relates the places under the CMA's management with the service at those places.
- Member_of identifies the group the user belongs to (for example, administrators or visitors).
- Essential defines requirements that the user can't negotiate or modify.
- OverLoaded lists interdependent requirements that can overload other requirements.
- Overridden defines requirements that will be preempted by essential or high-priority requirements.

Every user or service agent provides an initial profile with its requirements and preferences. The CMA uses the profile to set relationships and assembles the different instances into one composite component.

After composition, the CMA responds to the user or service agent with a CLS_Proposal ((2) in Figure 4b). The CLS_Proposal contains the template most closely matching the satisfaction criteria. If the agent accepts the CMA's proposed context specifications, it sends a CLS_Agree message ((6) in Figure 4a). If not, it reallocates the satisfaction point on the utility curve and sends a CLS_Repropose ((3) in Figure 4a) with the modified context parameters. The CMA then engages nearby CPAs and negotiates the new proposed context specifications by sending them a CLS_Request ((4) in Figure 4b). The CPAs collaborate to maximize the utility function satisfaction point. When they succeed, they send the new context parameters to the CMA through a CLS_Offer message ((5) in Figure 4c). The CMA responds by reproposing to the user or service, and the negotiation continues until they reach an agreement.

The agreement decision depends on one or both of these factors:

- The proposed context specifications max-

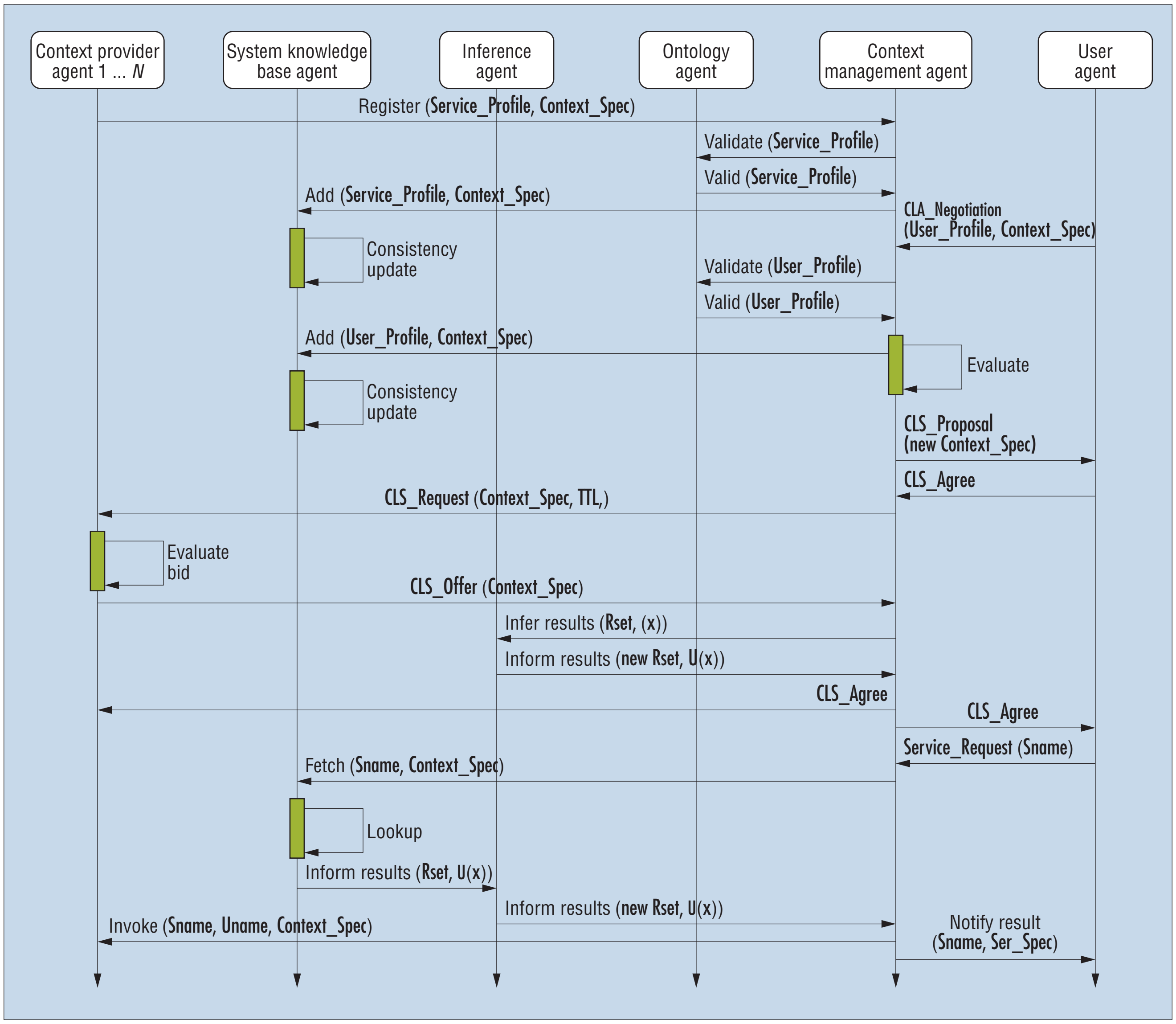
Context provider agent 1 … N
System knowledge base agent
Inference agent
Ontology agent
Context management agent
User agent
Register (Service_Profile, Context_Spec)
Validate (Service_Profile)
Valid (Service_Profile)
Add (Service_Profile, Context_Spec)
CLA_Negotiation (User_Profile, Context_Spec)
Consistency update
Validate (User_Profile)
Valid (User_Profile)
Add (User_Profile, Context_Spec)
Evaluate
Consistency update
CLS_Proposal (new Context_Spec)
CLS_Agree
CLS_Request (Context_Spec, TTL,)
Evaluate bid
CLS_Offer (Context_Spec)
Infer results (Rset, (x))
Inform results (new Rset, U(x))
CLS_Agree
CLS_Agree
Service_Request (Sname)
Fetch (Sname, Context_Spec)
Lookup
Inform results (Rset, U(x))
Inform results (new Rset, U(x))
Invoke (Sname, Uname, Context_Spec)
Notify result (Sname, Ser_Spec)

Figure 6. Message exchange between agents for registration and for negotiating context information.

imize the total gain in the utility function and can be provided by the CPAs.
- The negotiation deadline expires (in this case, the most recently negotiated context parameters are used).

Implementation

Our context-based conferencing environment customizes conference services using captured context and attendee and service profiles defined according to our context ontology. Defining CLAs for attendees using the negotiation protocol between CPAs and user agents greatly reduces the complexity of customizing the conference operation.

Figure 5 shows the services and agents in the conference environment. We divide available services into two categories:

- *Services in advance*, such as hotel booking, scheduling, and entertainment, take input context from attendee profiles provided during conference registration.
- *Services on the fly*, such as map and location, weather, and printing services, take input context from attendees' physical and logical contexts.

We implement agents using JADE, modifying the agent communication language (ACL) package to include our negotiation performatives while preserving JADE's FIPA (Foundation for Intelligent Physical Agents) compatibility for legacy applications.

The CMA provides APIs for registration, service selection, context specification, notification, and event-based triggering methods. The ontology agent provides APIs for browsing, navigation, searching, and validating ontology-based instances stored in a persistence repository and managed by the system knowledge base agent. The inference agent provides APIs for ontology-based inference such as subsumption, equivalence, similarity, and consistency checking. Moreover, it provides a fuzzy-based inference mechanism to support vagueness and uncertainty in context information such as location proximity, presence range, and service selection.

Our default deployment strategy is to install one CMA at each domain. Domains can be physical, such as a building, or logical, such as a network domain. We also deploy an inference agent at each domain, allowing it to clone itself on nearby hosts and form a federation in case of overloads.

Customizing a conference environment

Customizing the conference environment involves two phases corresponding to the two service categories. For services in advance, attendees log in to a Web page where they specify their context-based services requirements, such as hotel location, price, and length of stay. As Figure 6 shows, the user agent sends this information with the user profile to the CMA as a CLA_Negotiation message. The CMA registers the user and sends the user profile to the ontology agent, which validates it against the ontology constructs and rules. The ontology agent sends the profile to the system knowledge base agent, which adds it to the conference repository for further access and manipulation.

After registration and validation, the CMA sends the user's specifications to registered context providers to find matching services in advance. The CPAs reply with a CLS_Offer that shows available services. The CMA sends these results to the inference agent to find the closest match maximizing the user's utility function. Finally, after receiving the results from the inference agent, the CMA sends a CLS_Agree message to the user that includes the selected service.

The second phase—services on the fly—has the same message flow but, because the services are dynamic, shared, and distributed across the conference domain, agents can renegotiate context specifications, request available services, and infer received results at runtime. (In Figure 6, this process begins with the user agent's service request until the end of the message sequence diagram) Invoking services on the fly depends on current user context. Examples include invoking the printer that's nearest to the user's location and has the least load, and invoking a mapping service to provide directions, with the user specifying update rate and notification delay as the contextual specifications to be negotiated.

Experimental results

To evaluate our approach's feasibility, we deployed the agent system and conference application, interconnected via 802.11 wireless connections, at several locations in the University of Ottawa's Multimedia and Mobile Agent Research Laboratory. Conference participants mainly used laptops, PDAs, and stationary computers to access the system, which included a wide range of services, activities, and context requirements. The conference environment had one repository. We considered each physical location a separate domain and gave each domain one CMA, one ontology agent, and several inference, user, and service agents. Implemented agents were of moderate size to make them lightweight and easy to deploy in currently available handheld devices. User agents were 14 Kbytes, service agents were 16 Kbytes, CMAs were 21 Kbytes, and inference agents were 18 Kbytes.

Table 1. Context specifications used in negotiation.

Context specification	Preferred choice	Alterative choice
User location	Type: GPS → place name Delivery delay:5 ms Update rate: location changes	Type: GPS → location address Delivery delay: 10 ms Update rate: every 30 minutes
User availability	Available at: room name Update rate: location changes	Available at: hotel name Update rate: periodic
Services (printing)	Location: within 5 meters Availability: least loaded Capability: color, 10 pages per minute	Location: at most 20 meters Availability: medium Capability: 10 pages per minute
Media (wireless)	Availability: 75 percent Capability: 1 Mbps, 5 ms delay	Availability: 70 percent Capability: 20 ms delay

We conducted several experiments to evaluate the effectiveness of negotiating context. Effectiveness refers to the system runtime performance as the number of entities habituating the environment increases and resources remain constant. We measure this effectiveness by averaging both the delay in reaching an agreement to users' requests and the number of exchanged messages.

We used a range of context providers with different context parameters and service classes. For example, some context providers gave location and location-based information with parameters such as location granularity, update rate, and notification delay.

The number of conference attendees ranged from five to 100. Varying the number of users when calculating the average delay for negotiating context information let us measure the system's scalability. The average delay for 10 attendees was 861.75 milliseconds. Of this delay, 309.92 ms were for inferring and matching user requests and 243 ms were for subscription, repository access, and querying. We also counted the number of messages exchanged during the negotiation as indicating the overhead imposed by CLNP. We used these measurements as we expect these measures will be directly proportional to user satisfaction when the system is widely deployed.

Table 1 shows a typical context negotiation required by a conference attendee, including the initial preferred values and the alternative values if the system can't provide the exact requirements. Figure 7 is the actual negotiation message in OWL using the constructs defined in our ontology. It shows a negotiation for location, including the delay in providing location-based information. The message also includes the rule the user agent used to decide whether the CMA-provided value is acceptable: If the delay is between 5 and 10 ms, renegotiate.

Experimental results showed that the average delay in negotiating context is linear with a moderate number of users (about 60 percent of the system's expected capacity), becoming nearly constant as the number of users increases. CPAs tend not to renegotiate their proposed context specifications when they approach capacity, which reduces the number of messages exchanged and makes the average delay nearly constant. We found the same results with the number of messages exchanged, because the number of exchanged messages is directly proportional to average delay. The average number of messages exchanged at saturation point was eight, representing two round trips in the negotiation process.

We're extending the context ontology with fuzzy inference constructs to let agents reason with the uncertainty in captured context and to provide actions not explicitly defined in the context specifications. We're also adding an automated context-based inductive-fuzzy-inference mechanism to the inference agent. The mechanism will provide a dynamic CLA feature letting the CMA modify

```
<CLNP:CLNP rdf:ID="CLNP_khedr">
    <CLNP:hasRequester>
        <CLNP:UA rdf:ID="khedrAgent"/>
    </CLNP:hasRequester>
    <CLNP:hasContextSpec>
        <CLNP:ContextSpecification rdf:ID="Location"
            CLNP:hasContextElement="Location">
            <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
        </CLNP:ContextSpecification>
    </CLNP:hasContextSpec>
    <CLNP:hasProcess>
        <CLNP:NegotiationMessage rdf:ID="CLS_Proposal"
            CLNP:messageType="CLS_Proposal"
            CLNP:TimeToLive="20ms">
            <CLNP:hasRule>
                <CLNP:NegotiationRules rdf:ID="Rule1">
                    <CLNP:hasOperand rdf:resource="#and"/>
                </CLNP:NegotiationRules>
            </CLNP:hasRule>
        </CLNP:NegotiationMessage>
    </CLNP:hasProcess>
</CLNP:CLNP>
<CLNP:RuleOperand rdf:ID="and"
    CLNP:operandName="And">
    <CLNP:hasAntecedent>
        <CLNP:Antecedent rdf:ID="bigger_than"
            CLNP:hasValue="5ms">
            <CLNP:hasContextParameter>
                <CLNP:ContextParameter rdf:ID="DeliveryDelay">
                    <CLNP:usedByUtilityFn>
                        <CLNP:UtilityFunction rdf:ID="LogUF"
                            CLNP:hasSensitivityparameter="5">
                            <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
                        </CLNP:UtilityFunction>
                    </CLNP:usedByUtilityFn>
                </CLNP:ContextParameter>
            </CLNP:hasContextParameter>
        </CLNP:Antecedent>
    </CLNP:hasAntecedent>
    <CLNP:hasAntecedent>
        <CLNP:Antecedent rdf:ID="less_than"
        CLNP:hasValue="10ms">
            <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
        </CLNP:Antecedent>
    </CLNP:hasAntecedent>
    <CLNP:hasConsequent>
        <CLNP:Consequent rdf:ID="Renegotiate">
            <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
        </CLNP:Consequent>
    </CLNP:hasConsequent>
</CLNP:RuleOperand>
```

Figure 7. OWL representation of the negotiation message exchanged between UA and CMA.

some user requirements autonomously. Agents will no longer have to renegotiate when a given situation is similar but operates at a different location or time. One remaining challenge is to provide a communications mechanism spanning different agent domain platforms. Such a mechanism would provide a seamless, personalized, and completely context-aware pervasive environment. ■

References

1. M. Khedr et al., "Agent-Based Context Aware Ad Hoc Communication," *Proc. 4th Int'l Workshop Mobile Agents for Telecommunication Applications* (MATA 02), LNCS 2521, Springer-Verlag, 2002, pp. 105–118.
2. A. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, 2001, pp. 4–7.
3. K. Henricksen, J. Indulska, and A. Rakotonirainy, "Generating Context Management Infrastructure from High-Level Context Models," *Proc. 4th Int'l Conf. Mobile Data Management—Industrial Track*, 2003, pp. 1–6.
4. A. Richards et al., "Mapping User Level QoS from a Single Parameter," *Proc. 2nd IFIP/IEEE Int'l Conf. Management of Multimedia Networks and Services*, 1998.

The Authors

Mohamed Khedr is a PhD candidate in electrical and computer engineering at the University of Ottawa. His research interests include pervasive computing, agent-based middleware, context-aware environments, and semantic information modeling. He received his MS in electrical and computer engineering from the Arab Academy for Science and Technology, Egypt. Contact him at 800 King Edward Ave. Room 5105, mailbox 105, PO Box 450, Stn. A, Ottawa, ON K1N 6N5, Canada; mkhedr@site.uottawa.ca.

Ahmed Karmouch is a professor of electrical and computer engineering and computer science at the University of Ottawa's School of Information Technology and Engineering. He also holds an industrial research chair from the Ottawa Carleton Research Institute and Natural Sciences and Engineering Research Council. His research interests are in distributed multimedia systems and communications, mobile computing, home architecture and services, context-aware ad hoc communications, and mobile software agents for telecommunications. Karmouch received his PhD in computer science from the University of Paul Sabatie,. He's a member of the ACM and the IEEE. Contact him at 161 Louis Pasteur St., Univ. of Ottawa, CBY, Ottawa, ON K1N 6N5, Canada; karmouch@site.uottawa.ca.