Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Network and Computer Applications 28 (2005) 19–44
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

ACAI: agent-based context-aware infrastructure for spontaneous applications

M. Khedr*, A. Karmouch

Multimedia and Mobile Agent Research Lab, School of Information Technology and Engineering, University of Ottawa, 161 Louis Pasteur St. Ottawa, Ont., Canada K1N 6N5

Received 27 June 2003; received in revised form 30 April 2004; accepted 30 April 2004

Abstract

The vision of people amalgamated with their surroundings in a spontaneous way created a new, context-aware era of human-computer interaction. The goal is to provide an infrastructure that can understand the current situation and act on that understanding. Context-awareness means that people, services and artifacts in an environment are integrated in a homogeneous manner in order to provide seamless service while still preserving privacy. To evolve from a passive state to an active pervasive state, the infrastructure must be able to support uniform context representation, to reason about context, to offer context-based service discovery, and to support a context management and communication protocol. In this paper, we present ACAI, an innovative Agent-based Context Aware infrastructure, equipped with the capabilities required to maintain spontaneous applications both locally and across different domains. We describe our ontology for modeling context that provides a common understanding of what context means and facilitates context inference. We propose a multi-agent framework that parallels the infrastructure design in order to assist in the development and runtime provisioning of spontaneous applications.

Keywords: Context aware; Ontology; Context service discovery; Context inference

1. Introduction

Pervasive environments (Davies and Gellersen, 2002) are characterized by a wide range of devices and resources that use heterogeneous networks and perform a variety of

Abbreviations: AMS, agent management system; ACAI, agent-based context-aware infrastructure; CMA, context management agent; CPA, context provider agent; CLA, context level agreement; DF, directory facilitator; OA, ontology agent; SIP, session initiation protocol; SA, service agent; UA, user agent.

* Corresponding author. Tel.: +1-562-5800; fax: +1-532-5174.
E-mail addresses: mkhedr@site.uottawa.ca (M. Khedr), karmouch@site.uottawa.ca (A. Karmouch).

1084-8045/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2004.04.002

tasks in spontaneous ad hoc communications. These dynamic environments need an infrastructure that proactively provides the participating entities with a rich set of capabilities and services all the time, everywhere, and in a transparent, integrated and extensible way. Context (Dey, 2001; Dey and Abowd, 1999) plays an important role, providing information about the present status of people, places, things and devices in the environment. Context includes location, time, activities, and the preferences of each entity. However, the design and development of spontaneous applications introduce new requirements and challenges not found in conventional applications. On the one hand, dynamic and adaptive provisioning requires an expressive, semantically rich representation to support context information, (location, identity and activity), context engagement (required by certain events to trigger actions), and context dependency (the relationship between different aspects of context information). On the other hand, seamless concepts require a service layer in the infrastructure that is capable of delivering functionalities such as context management, context-based service discovery, and a communication protocol responsible for handling issues such as presence, notification and privacy.

Our Agent-based Context-aware Infrastructure (ACAI) is a new infrastructure designed to support these requirements. ACAI integrates the different physical spaces of the active users and provides them with context-based core services in a transparent fashion. Spontaneous situations are those arising from natural interactions; they are controlled and directed internally, localized, and handled without previous planning. ACAI addresses the following requirements:

- Physical and medium abstractions.
- Unify context representation.
- Context composition, inference and dissemination.
- Context-sensitive communication protocol.
- Context management.
- Programming interfaces for application development.

Fig. 1 presents our vision of such an environment, depicting a typical world of a mobile user. The ***home site*** represents the user's principal physical space, normally the office or home. The ***home site*** contains the user's personal context information such as schedules, profile and activities. The ***current site*** is the present physical location, where the user performs activities such as a making presentation in a meeting. The user interacts with the surroundings, uses services found in ***current*** and ***local*** sites, and has social activities with others. He might be also engaged in activities that occur in a ***remote site***, such as a video conference.

A typical scenario for ACAI is the day of a University professor, who flies to a new city, and stays the night for two project meetings the next day, one in the morning, and one in the afternoon. He has a cell phone, and occasional access to the Internet through his PDA and laptop. Without ACAI, he is late for his first meeting because of traffic delays he did not know about. During the meeting, he cannot answer some questions because of a lack of connection, or a frequent disconnection between his laptop and his office at home. The second meeting is postponed due to weather conditions, and is re-scheduled for the same time the next day. Unaware of this change in schedule, the professor goes to the location of

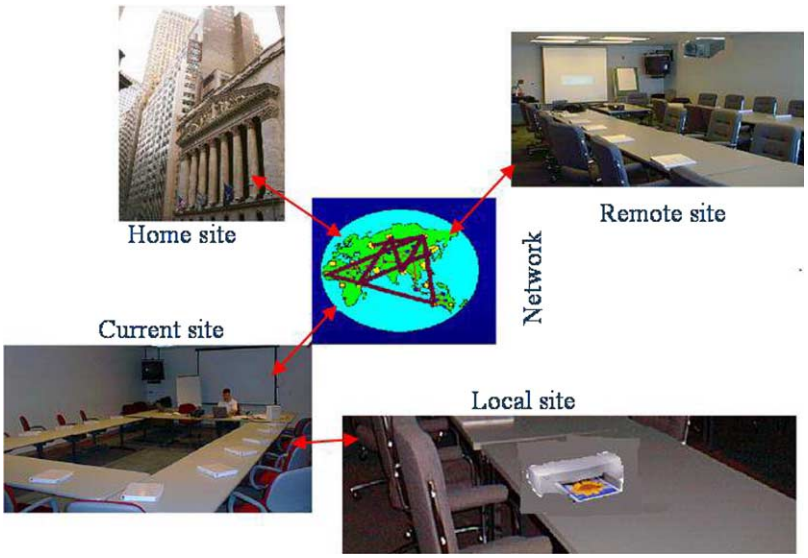

Fig. 1. The ACAI environment.

the other meeting and feel frustrated with the wasted time. Away from home for another day, he has trouble finding city information because the PDA and laptop has no consistent network connections for most of the day.

With ACAI, the professor's PDA has a connection with the local weather and traffic network; his agent matches weather and traffic information with his calendar, and, based on matching algorithms, sets the wake-up call 15 min earlier. His cell phone provides the best route, allowing him to arrive at the meeting on time. Since his PDA is in seamless connection with his home desktop, he makes changes to his slides, and adds backups, while in the taxi. He forgets to turn off his cell phone in the meeting, but ACAI knows to let it ring only in an emergency. During the meeting, his PDA provides the information he needs, and generates menus with simple, quickly selected options. He is able to answer questions, and to send slides to the PDAs of other participants. In the first meeting, the agent receives the information that the second meeting has been postponed, and arranges for another night's stay. It connects to the city information site and suggests recreation activities to occupy the rest of the professor's day. These spontaneous situations clearly require an infrastructure that can provide core services, and can reason about current situations.

The paper is organized as follows. Section 2 presents the ACAI infrastructure, its layers, and the functionalities provided by each layer. Section 3 introduces the multi-agent system that makes the functionalities possible. Section 4 discusses our new ontology for context representation, and our proposed inference mechanism based on semantic logical inference, constraints satisfaction and context-dependency conditions. Section 5 introduces the core ACAI services, including the service discovery that provides seamless resource engagement, and the context-sensitive communication protocol that allows communication between entities. We describe a typical ACAI scenario, and evaluate

a prototype in Section 6. Section 7 presents related work, and Section 8 concludes the paper with ideas for future work.

2. Agent-based context-aware infrastructure (ACAI)

ACAI is an infrastructure that allows context information to be collected, processed, inferred, and disseminated to spontaneous applications. It provides this interaction seamlessly, without revealing the inherent complexity required to manage the heterogeneous sources that provide the context information.

All this is achieved through a layered architecture, as shown in Fig. 2. In the first layer, context is sensed and captured through sources embedded in the environment. In the second layer, the context is interpreted and structured by the context ontology module. The context inference module deduces other contextual information that has not been explicitly sensed by the first layer. Context is now ready for invocation by applications and mobile users. However, ACAI provides additional functionalities such as service discovery that provides awareness about services in the environment. An example might be a printing service to accommodate the user's location and printing preferences. It provides context management so that context can be stored, queried and accessed by users and resources with limited capabilities. It also provides a context-sensitive communication protocol for event notification, service and information delivery, and presence projection. These value-added functionalities are passed to mobile users and applications from the third layer. This layer functions primarily as an interface to the lower layer, allowing context information to be negotiated with the context providers, and for context level

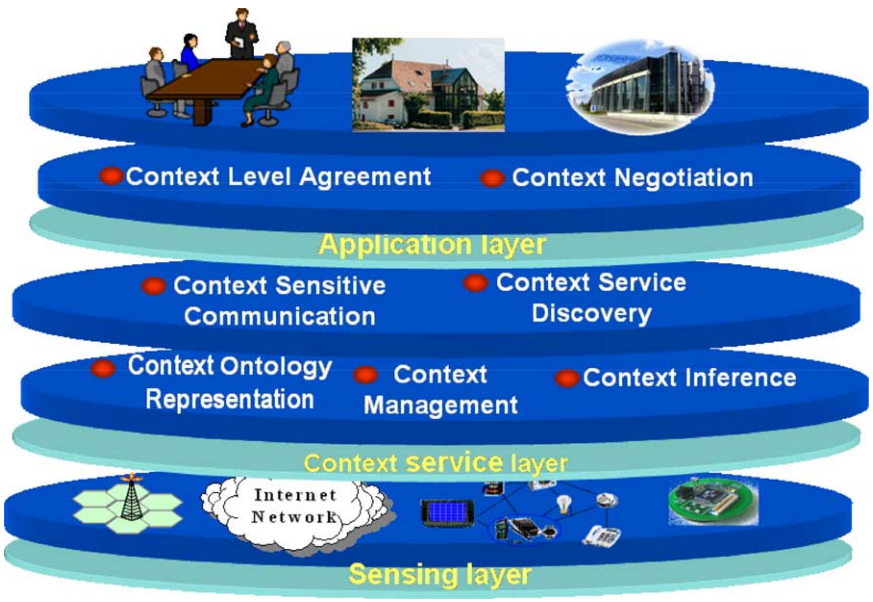

Fig. 2. The ACAI layered architecture.

agreements to be made between different domains. The functions of the three ACAI layers are summarized below.

2.1. The sensing layer

The sensing layer is responsible for capturing and acquiring context information about the elements in the environment. It consists of sensors, agents and network probes that co-operate to monitor context information, which is then passed to the context service layer.

2.2. The context service layer

The context service layer has two goals: (a) to represent context in a well-structured, unified, composable and extensible model, and (b) to provide context-based services. The first goal is achieved by using an ontology model for representing context, and a context inference module. The inference module examines context in order to extract and define new context information that is undetectable by current sensors. It can, for example, infer contextual relations and dependencies. The second goal is achieved through three modules: the context management module, the context-sensitive communication module, and the context-based service discovery module. The context management module is beyond the scope of this paper and will be discussed in further work. The context-based service discovery module provides services and resources through a peer-to-peer discovery process in which each peer node holds information about services, users, and even context of interest to each peer. For seamless access, spontaneous applications need to interact with existing devices, services and users, and therefore to communicate with them after their discovery. This is achieved by the context-sensitive communication module that uses unique identifications via application-level addressing. This module provides presence, notification, and network connectivity within the same domain and across different domains.

2.3. The application layer

The application layer provides the interface between mobile users and applications on the one hand, and the ACAI context service functionalities on the other. Two processes are implemented in this layer: (a) the context level agreement (CLA) and (b), the negotiation of context specifications so that the agreed context information can be achieved (Khedr and Karmouch, 2004). CLA (Khedr and Karmouch, 2003) defines which levels of context are of interest to end-users and attainable by context providers. It has the following unique advantages:

- It relieves end-users from the need to deal with context interpretation, management and reasoning.
- It protects context providers against unexpected degradation in performance, as they are aware of the context required by entities in the system.
- In large-scale environments with many domains, CLAs are used to provide seamless context-based service assurances.

– It offers an easy mechanism for context providers to intelligently provide and monitor context and resources.

We defined three broad types of CLA. The first is the passive context level where contextual requirements are simple, and need only simple operations from the context provider. The second is the active level where users are more interactive; their context specifications are a result of complex operations on the part of the context providers, such as filtering, merging, composing and monitoring. Negotiation at this level (Khedr and Karmouch, 2004) is limited to the satisfactory provision of those operations. The third level is the dynamic level, where context management and inference are essential in order to provide higher context levels. Negotiation at this level aims to provide personalized context such as storage, delivery, remote access, and inference. For a CLA to be dynamic, automated and extensible, it must negotiate context specifications through an autonomous agent system representing entities in the environment. Further discussion about the CLA and negotiation may be found in another paper (Khedr and Karmouch, 2004).

3. ACAI agent model

ACAI's agents are designed and developed as an extension to our work on integrating an agent paradigm in context-aware environments (Khedr et al., 2002). The Context Management Agent (CMA), the Coordinator Agent (CA) and the Ontology Agent (OA) are considered the heart of ACAI's multi-agent system. The Reasoner Agent (RA) and the System Knowledge Base Agent (SKBA) are optional; they are required in certain situations, according to specific application requirements. The CA is responsible for managing information located in the repository space. It interacts with the SKBA in order to assign, load, modify and delete information stored in the repository space according to policies supported in ACAI. The CMA is responsible for managing ACAI's open sessions and the environment's resources. The purpose is to achieve fairness in resource allocation, and guarantee the context level negotiated during session initialization. The CMA also monitors changes in context to detect any violations in agreed context levels, and to resolve any conflicts that may occur. The SKBA wraps the repository associated with the agent system. It stores ACAI's policies, user profiles, service invocation conditions, and agreed contextual information during the setup negotiation, and semantically matches and retrieves stored information and ontologies.

This section discusses the aims and roles of each agent. The main objectives of our design are management, coordination and semantic interoperability. We therefore have three core agents: the CMA, the CA and the OA. These will be discussed in detail, followed by a discussion of the value-added agents: the RA and the system knowledge base agent.

3.1. Context management agent (CMA)

The CMA is the administrator of the system, responsible for monitoring on-going context-based sessions and managing the environment resources. The CMA also has the authority to cancel, modify and renegotiate context specifications according to changes that

might occur in the environment, or if agents violate the tasks assigned to them during negotiation. Agents register with the CMA both to project their presence to other agents in the environment and to 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. This repository is used to retrieve information about entities in the environment, and for the temporary storage of session information. The CMA accepts subscription requests from user, service, and context provider agents to be added to the agent directory facilitator. This subscription phase allows the CMA to search and select the best available context that will satisfy the agreements required by applications and users. In addition, the CMA binds the CPA to applications according to their context agreement. It stores this information in the system knowledge base for future use, so that resources can be reconfigured as the context changes.

3.2. Co-ordinator agent (CA)

The CA is responsible for managing the entity-capsule space, the temporary repository space in the ACAI. The CA supports the following functionalities:

- Active coupling between capsules that have context information in common.
- Anonymous offline coupling between entities and their associated capsules.
- Adaptability to cope with frequent changes in the environment itself, or in the contextual information of the entities in the environment.
- Comprehensive selectivity for deciding the relevant information to be retrieved from the knowledge base and to be stored in the capsule space.
- Dynamic capsule allocation for fine control over the available capsule space and for altering the information stored in the each entity's capsules to better represent its context.

The main reason for a temporary repository space in ACAI is to avoid frequent information exchange between the agents and the system knowledge base. A good level of performance and interoperability is therefore possible, even when a huge amount of information is exchanged. Another reason is that domain knowledge about a specific single entity is easier to establish than the domain knowledge of a complex system. It is therefore simpler to identify and implement contexts on entity level than on system level. In ACAI, every active entity will have at least one entity-capsule that holds relevant information from its profile and its current context. This may raise the question of how this information is determined. Our system does so through the context level negotiation protocol (Khedr and Karmouch, 2004). The CA is responsible for resolving conflicts arising from new facts and rules made by the RA. The CA is also responsible for runtime validation and consistency checking between different capsules. It further ensures that changes made to one entity capsule as result of reasoning by the RA do not result in a violation or inconsistency in other capsules. Two methods are used to resolve conflicts:

- *Semantic resolution*: the CA invokes the OA in order to compare the new context with the ontology schema. This ensures consistency with the defined concepts and their properties.

- *System resolution*: the CA invokes the CMA to check if modification in the information made to one of the entity capsules will affect other situations, or cause violations to agreed-upon context levels.

3.3. Ontology agent (OA)

The OA provides the semantic capability of ACAI. Implemented using OWL, it uses ontology engineering to model context and system information. At negotiation time, it formulates context specifications from agents' requests and profiles according to the previously-designed ontology constructs, axioms and rules of composition. This information is then provided to the CMA for further processing. The OA also provides searching and browsing interfaces so that registered agents can look up and use available ontologies. In addition, the OA is responsible for the conversion of raw context data to their OWL representation. This is subject to further processing such as validation, consistency checking, and modification according to ACAI policies. ACAI provides every client's device with a compact cloned version of the OA, called the personal ontology agent (POA). The POA has built-in behavior that generates a semantic representation of user information. Every POA registers with the main OA in the ACAI server for reasons of reliability and presence. If ontology constructs are needed, such as the modeling of new capabilities or the preferences of a device, the POA requests the constructs from the main OA, achieving interoperability and context provisioning on demand.

3.4. Reasoner agent (RA)

The RA contains a new hybrid inference system that integrates logical reasoning, fuzzy reasoning and semantic rule representation into one system. In the context inference process, context captured from sensors and from users' and services' profiles is treated as a series of facts. The RA uses these facts (stored in the system knowledge base repository) to deduce new context information and to trigger actions. The RA uses logic-reasoning mechanisms to ensure that instances of captured context are consistent both with each other and with arguments defined in the ontology. This allows the construction of an inferred hierarchy of contextual information based on the ontology classes. A priority resolution mechanism is used to deal with conflicts and inconsistencies that may arise from firing rules and triggering actions.

3.5. System knowledge base agent (SKBA)

The use of ontologies requires a storage facility and a facility manager. This management role is taken by the SKBA. More precisely, it acts as the interface to the repository where queries for stored information are sent, and responses are returned. The repository also includes tables for storing the information that links context providers and the entities requesting the information. These tables record response time, the number of messages exchanged in open sessions, lists of registered agents, and their roles. All these tables are managed and manipulated through the SKBA.

3.6. Context provider agent (CPA)

Each context source is wrapped with a context provider agent that is responsible for capturing raw data from the source and interpreting it. The CPA involves the OA in the mapping between the raw context format and its corresponding OWL ontology. This information is stored in the system knowledge base so that other agents can use the look-up service provided. Furthermore, the CPA is responsible for negotiating context specifications acquired by the context source under its control, and for monitoring ACAI sessions it is currently involved in.

4. The context ontology model

Research in context-aware applications (Henricksen et al., 2003; Román et al., 2002; Lum et al., 2002; Yau et al., 2002) defines models for context that are based on a top–down architecture. Context is modeled according to the tasks the application layer performs and not according to its functional intentions. These models are ad hoc in nature: the problem lies in their extensibility and their interoperability with other systems. Ad hoc modeling of context also limits the use of the system across different domains due to the lack of a common understanding of the meaning of context information.

Ontologies represent a new paradigm for modeling and representing the context needed in pervasive computing. An ontology provides the information about the 'what, where and when' needed for spontaneous applications. During the ontology design, we faced the question of what contextual information to model. This was largely due to the overwhelming variety of possible context. We concluded that to completely formalize all context information is unrealistic. Instead, we found that it is more appropriate to model context in the form of **levels of expressiveness**, shown in Fig. 3a, where the highest level is an abstraction of all concepts of context. As we go down through the levels, context is expressed in more detail and refined more concretely.

The highest level of abstraction is the ***ContextView.*** This provides a point of reference for declaring context information. ***ContextView*** represents the different types of context that belong to a given entity. An entity could be a user, a service or even the environment

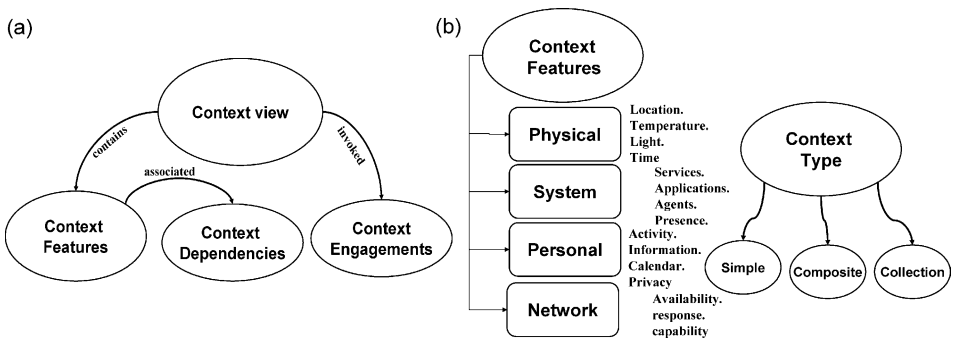

Fig. 3. (a) The upper context ontology. (b) The defined context features and the associated context types.

itself. At least one instance of the ***ContextView*** will exist for each entity in the environment. The properties ***contains***, and ***invokes*** are properties of the ***ContextView***. The classes ***ContextFeatures*** and ***ContextEngagements*** are the respective ranges of those properties. These classes are considered to be the second level of expressiveness in the ontology. Each instance of the ***Contextview*** will contain a subclass of the ***ContextFeatures*** that is ***associated*** with one or more ***ContextDependencies*** classes and is ***invokedby*** a ***ContextEngagements*** class.

The ***ContextFeatures***, shown in Fig. 3b, consists of the classes related to the physical environment (such as location), classes related to the system (such as services), classes related to persons (such as ongoing activities and identity) and classes related to the underlying network (such as availability and bandwidth.) Each of these classes is categorized to be either simple, a composite of other context information, or a collection of similar context information. The current version of the ontology, Fig. 4, defines the following five context elements in the ***ContextFeatures***:

4.1. The location element

The location element is a subclass of the physical ***ContextFeature***. It provides information about the location of entities as identified and detected by location-type sensors. Location information currently includes concepts such as a university, a company, and a generic place that developers can extend. The location element represents location with two main classes. The ***ContextualLocation*** class is used to represent the low-level location information captured from sensors in the environment. The ***PlaceInstance*** class is used to represent high-level location information. The former is usually used by ACAI to cope with the different technologies that represent location information, and to map

- ContextFeature ()
  - ActionProfile (actionEndTime, actionExpectedDuration, actionStartTime, actionprofileName, hasInvolvingAgents, hasInvolvingPersons, hasInvolvingService)
  - ComplexityCase ()
    - Collection (groupedOf, hasSize)
    - Composite (composedOf, hasSize)
    - Simple (partOf)
  - Network (hasNetworkProfile, nDomain, nName)
    - NetworkResource (hasServiceProfile)
      - Emailserver ()
      - LapTop ()
      - PDA ()
      - Phone ()
      - Printer ()
      - WebServer ()
    - NetworkTopology ()
      - Wireless (hasAccessPoint, hasAccessPointAddress)
        - 802.11 (modeOfOperation)
        - Bluetooth (modeOfOperation)
      - Wireline ()
        - Ethernet ()
        - Internet ()
  - NetworkProfile ()
    - NetworkProtocol (hasProtocolChar, protocolName)
    - NetwrokParameter (nParameterName, nParameterValue)
    - ProtocolCharacterisitic (PValue)
  - PersonProfile ()
  - Physical ()
    - Actors (actorID, actorLocation, actorName, actorRole, hasCurrentAction, usesCurrentService)
      - Agent (hasCommunicationLanguage, hasPlatformType)
      - Person (hasPersonalProfile)
      - Thing ()

- Location (altitude, degree, direction, hasComplexityCase, hasCurrentAgents, hasCurrentPersons, hasLoadedServices, hasLocationRange, hasNetworkSupport, hasRestrictedActions, latitude, longitude, minute, placeName, second)
  - Company ()
    - InsidePlace ()
      - Building ()
        - ActivityPlace ()
          - FoodPlace ()
          - LibraryPlace ()
          - SportComplex ()
        - Floor ()
        - Room ()
          - GuestRoom ()
          - MeetingRoom ()
          - Office ()
        - Section ()
          - Departement ()
          - ReceptionHall ()
    - OutsidePlace ()
      - ActivityPlace ()
        - FoodPlace ()
        - LibraryPlace ()
        - SportComplex ()
- Place ()
  - InsidePlace ()
    - Building ()
      - ActivityPlace ()
        - FoodPlace ()
        - LibraryPlace ()
        - SportComplex ()
      - Floor ()
      - Room ()
        - GuestRoom ()
        - MeetingRoom ()

Fig. 4. A partial representation of the context ontology classes and associated attributes.

location-related information that is similar at the lower level, but not at the high level. The latter is used to project location information in a suitable format that is understandable by entities using ACAI.

4.2. The actor element

The actor element is a subclass of the physical ***ContextFeatures***. It provides information about the identities of the entities in the environment that are detected using an identity context provider such as RF tag. The Actor element has two main subclasses, the person class and the agent class that represents the software agents involved in the system. The identity information includes the ID value, the type of technology used (such as an infrared model or an RF tag), the role played by this entity (such as a printer, a user), the time of detection, and a reference to the actor's personal profile for further context acquisition.

4.3. The network element

The network element identifies concepts such as network resources, network protocols and topology. We limit our scope to wireless networks, especially the 802.11 and Bluetooth. Network resource such as routers and access points are each represented by a service profile containing their capabilities and preferences.

4.4. The service element

The service element represents services discovered in ACAI environment. In the ***ContextFeatures*** system, the class has entries for the service profile, service parameters, service complexity, and the applications engaging the service.

4.5. The action element

The action element represents the social activities that users encounter in their current site either with other users or with the services detected and projected to their context view. Activities currently defined in the ontology are limited to the scenario we present in the implementation section. This includes audio and videoconferencing, a presentation made either locally or remotely, and regular activity such as emailing, chatting and files sharing. The complete ontology can be found in http://www.daml.org/ontologies/337.

The professor in our Introduction, for example, would have context information as described below:

- An identification
  < Context:Person rdf:ID = "Joe Doe" >
  < Context:actorRole rdf:resource = "#Professor"/>
  < /Context:Person >
- A phone number
  < Context:ServiceParameter rdf:ID = "PhoneNumer" >
  < Context:sParameterValue >

< rdf:value > 6195 < /rdf:value >
< /Context:sParameterValue >
< /Context:ServiceParameter >

– When the professor is in his office
< Context:Office rdf:ID = "B502" >
< /Context:Office >
– The professor's first meeting is a tutorial about his current research. He uses the printer available at that location which has a 100MB Ethernet network connection.
< Context:LocalMeeting rdf:ID = "Tutorial " >
< Context:actionName >
< XMLSchema:string >
< rdf:value > Presentation </rdf:value >
< /XMLSchema:string >
< /Context:actionName >
< Context:usesCurrentService rdf:resource = "#HPLaserJet"/>
< Context:usesCurrentService rdf:resource = "#Internet"/>
< Context:hasCurrentPersons rdf:resource = "#Joe Doe"/>
< Context:hasCurrentPersons rdf:resource = "#Peter "/>
< Context:usesCurrentService rdf:resource = "#Mitel_IP"/>
< /Context:LocalMeeting >
< Context:Ethernet rdf:ID = "100M_Ethernet" >
< /Context:Ethernet >
– The printer service profile has the following attributes
< Context:ServiceProfile rdf:ID = "HPLaserJetProfile" >
< Context:hasNetwrokSupport rdf:resource = "#100M_Ethernet"/>
< Context:hasServiceLocation rdf:resource = "#B502"/>
< Context:hasServiceParameter rdf:resource = "#Color"/>
< Context:hasServiceParameter rdf:resource = "#PaperSize"/>
< Context:hasserviceActor rdf:resource = "#PrinterAgent"/>
< Context:hasServiceParameter rdf:resource = "#Resolution"/>
< /Context:ServiceProfile >
< Context:Agent rdf:ID = "PrinterAgent" >
< /Context:Agent >

4.6. Context inference

One of the major difficulties facing context-aware applications is acquiring and interpreting undetectable context information. While a user is easily detected by sensors, that user's social relation, colleagues, and priority constraints are nearly impossible to detect. In order to overcome this difficulty, ACAI must be able to reason about the context information once it is gathered from the sensing layer and modeled using the ontology. We therefore developed a relational and dependency ontology model and implemented an inference engine in order to derive logical, social and composable context from the context initially captured. The proposed dependency ontology uses prolog syntax to facilitate the integration of the inference engine and logic server. In developing the structure, we used

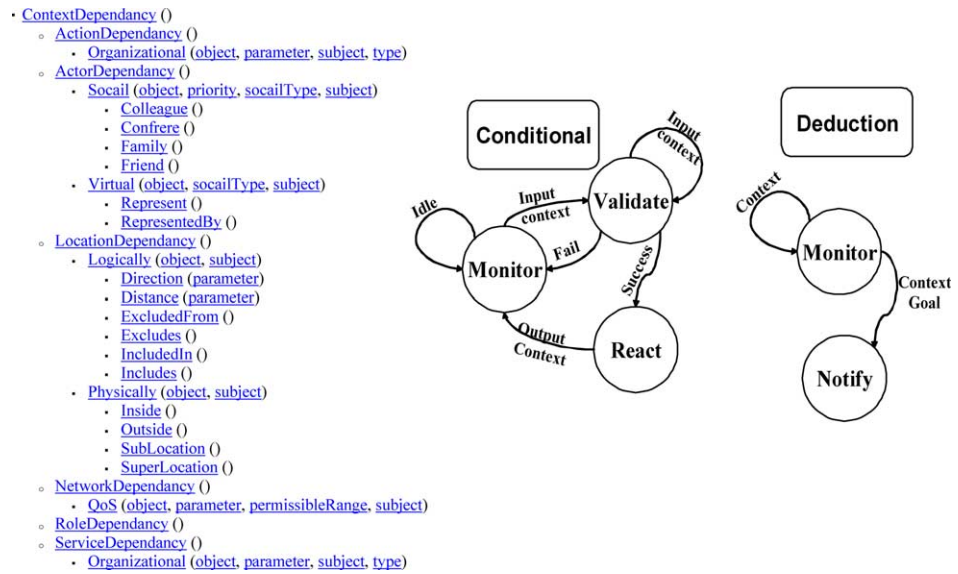

Fig. 5. The context dependency ontology and the context inference state diagram.

the same concept of ***levels of expressiveness***. The structure is composed of a predicate with one or more arguments that define the context used. Context from the user and service profiles in the sensing layer are treated as facts in the context inference model. With these facts, the inference engine builds a rule-based knowledge server that it uses to deduce new context information. The dependency ontology, shown in Fig. 5, has a wide range of predicates that correspond to the different ***ContextFeatures*** represented. The ontology consists of five rule-type categories.

- The ***ActionDependency*** predicate infers and deduces context information related to actions performed by users and service. The arguments called ***Object*** and ***Subject*** represent the two entities that have an action relation. The ***parameter*** argument represents the relation value and the ***type*** argument represents the triggering condition value. For instance, the rule *AudioConference (User a, User b, present, 2:00 PM)* means that for an audio-conference activity to start, user ‘a’ and user ‘b’ must simultaneously exist at 2:00PM.
- The ***ActorDependency*** predicate deduces social information such as the relation between users. For instance, the following rule infers the relation between two users at a certain place to be an employer–employee relationship with a 100% certainty factor. Social(user a, user b,Employer,100): Employee(user a), Employer(user b), Place (Company x)
- The ***LocationDependency*** predicate provides context information that allows location-based information to be deduced. This may be a place that is part of another place, and located in a certain direction and within a certain distance.
- The ***ServiceDependency*** predicate provides relationships between services that can be invoked and composed at runtime.

- The ***RoleDependency*** predicate complements the action and service dependency by providing context information about activities and roles played by the entities in the environment.

After developing the ontology, the next step is the design of the inference engine, in the form of a state machine engine. The state diagram, shown in Fig. 5, is used to control the operation of context reasoning based on the input information and the dependency predicates. The inference engine supports two methods of reasoning: the conditional method and the deduction method.

4.6.1. Conditional inference method

For many context-based applications, it is not possible to identify all possible situations beforehand, nor to have the system perform the right actions as a result. To overcome this difficulty, the conditional inference method uses an algorithm that is a variant of analogical reasoning, combining analogical and non-monotonic reasoning. The inference engine keeps track of the current context and looks for the rules that are satisfied. It then matches the arguments of the rule with the available context and implements the corresponding actions. If the match is not exact, it uses the one that matches known context best, and implements the actions associated with it. It then adds the new deduction to the monitoring phase and starts its reasoning again. For example, in our remote videoconference, the system might capture context about available users, applications to be shared, and services to be invoked. The engine then uses the rules and the dependency predicates to keep track of the current context information, and looks for situations that will move the inference from validation to reaction. If successful, it produces new context information, such as running a presentation or dimming the lights in the room. If the context is altered, the inference engine revaluates its state and invokes the actions appropriate to the new situation. This conditional inference is achieved in the following way:

- Checking whether context captured by sensors is consistent (a) in itself and (b) with arguments defined in the context dependency model.
- Constructing a hierarchy of the context information on the basis of the context ontology classes.
- Looking up the rules most specific to the given situation.
- Checking whether firing a certain rule may conflict with other situations, and if so, proposing alternatives.

4.6.2. The deduction inference method

The deduction method is used to decide the most accurate action from many alternatives. The inference engine uses context in conjunction with available dependency predicates as facts and compares them with the implications stored in the knowledge base to deduce new facts. For instance, in our scenario with the professor:

An implication: If Professor starts the presentation then connect his laptop to his home desktop.

An Axiom: The Professor started the presentation.
Conclusion: Connect his laptop to his home desktop.

This conditional inference is achieved in the following way:

- Finding the minimum set of actions for which a given set of context is correct.
- Checking whether firing a certain rule may conflict with other situations, and if so, proposing alternatives.
- Enforcing global behavior and policies.

5. ACAI core services

5.1. The context based discovery module

The context based discovery module provides a unique method by which applications, services and devices can discover and invoke other applications, services and devices according to their capabilities, to their context information and through inference. The module also enables applications, services and devices to project their existence, and to establish interoperable peer-to-peer sessions with other entities.

Context providers in ACAI infrastructure and the inference engine act as the context brokers for the mobile users and applications. Context providers capture context about services and users in the proximity and publish the information to other entities in the environment. Each entity will have its own current service directory, which contains information about services and users that match its needs in its current context.

Entries in the directory are transient and change according to the context of the user. When a user requests a service, it is looked up in the directory. If nothing is found in the local directory, an ontology-based query is sent to ACAI manager. ACAI manager then formulates a dependency rule and sends it to the inference engine. The dependency rule acts as a subscription method, asking to be notified about the existence of the service. The inference engine will use the deduction method and notify the requestor of the result. The inferred output is either that the required service exists, in which case the information is sent to the user, or that it does not exist for the time being. The inference engine then switches to the conditional method to monitor context for the possibility that the service may exist in the future.

Entries in the local directory are automatically updated and modified by the information from the context providers. The service discovery algorithm consists of two phases, the context discovery phase and the context association phase.

- Context discovery phase

ACAI considers each user, device or application to be a service entity which can be detected, engaged, and monitored by other entities. Context providers collaborate with the inference engine and advertise the context of the entities in the environment. They tend not to advertise the service itself as in existing standards, but rather its context. The strength of this concept is that by knowing an entity's context, others are able to query it autonomously, and to discover its capabilities and the way in which it is invoked. This is

in contrast to traditional service discoveries that only provide a description of a service in a context—insensitive manner. For instance, instead of letting the printer advertise itself, as in case of SLP or salutation, the discovery is delegated to the context providers that detect the printer's existence. Context providers will then advertise the existence of a printer with its context such as location, availability and status. Discovery delegation also has the advantage of efficiently adapting to changes in the environment, the main goal of ACAI infrastructure. Entities receiving the context information will know about the existence of new services in the vicinity, or changes to previously discovered services. Using the published context, peer entities can build their own service directory, reflecting their personalized view of the environment. The directory is inherently reliable and automatically maintained: context is naturally and continually published by context providers. When context is missed or mistakenly ignored on one occasion, the situation is corrected on the subsequent occasion.

- Context association phase

The context discovery phase is suited for physical, network and system features but falls short with respect to social and dependency context. This limits discovery to apparent context and ignores implicit context that can be inferred through our proposed inference model. Context association allows discovery services according to the dependency ontology. Entities provide ACAI with their service or user profile, including relational, dependency and sensitive context such as privacy parameters. These profiles, along with the system profile that includes policies for authentication, access rights and management (Harroud et al., 2003), are fed to the inference engine as facts and rules. The inference engine builds the knowledge base with these predicates and triggers relational and dependency actions that satisfy these rules.

5.2. The context sensitive communication protocol (CSDP)

ACAI's second service is a communication protocol sensitive to frequent changes in the environment. This is a context-sensitive protocol capable of creating, modifying and adapting sessions while still maintaining the users' privacy preferences (Khedr and Karmouch, 2003). We have identified five characteristics that a context-sensitive communication protocol must handle.

- *Mobility*: CSCP must support inter-domain and intra-domain mobility. It should also support distributed, peer-to-peer, and event-based computing. This mobility must not come at the expense of persistent connectivity and seamless accessibility to resources.
- *Management*: CSCP should support session management for real-time and non-real-time applications. Management includes soft handover between different domains, and network abstraction to users and applications. This requires state-full and stateless binding mechanisms and methods to notify presence.
- *Technology transparency*: CSCP should be independent of standard technology and devices as much as possible. This will ease extensibility and interoperability between different networks and administration domains.
- *Association*: CSCP must associate services or project presence to other users based on context captured from the environment, or from user profiles stored in the system

knowledge base. This facilitates spontaneous invocation of services and presence-based communication.

- *Personalization*: CSCP must provide seamless service provisioning geared to user requirements, supply familiar interfaces, and have a storage capability that takes advantage of the common understanding of context.

To use and invoke context, agents must first locate context provider agents, then engage in a communication process with them and with the other agents currently available. An initial system setup and methods for seamless communication are required. For this purpose, we extend the Session Initiation Protocol (SIP) (Schulzrinne and Rosenberg, 2002). SIP is an IETF signaling protocol that defines how to establish, maintain and terminate collaborative multimedia sessions. It provides mechanisms so that User Agents and Proxy/Redirect servers can determine the current location of a particular user and perform call setup. This includes Session Description Protocol (SDP) for accurate media description. When a user moves from one location to another, he sends a REGISTER request via the SIP-UA to the SIP Proxy with his new location or his expected destination. When a call is performed (INVITE), the server determines the current location of the callee so that a connection can be established with the caller. We have developed a wrapper agent (W-SIP) that interfaces SIP-UA to the NSA allowing automatic availability and presence. We integrate SIP into ACAI's architecture because of its ability to support mobility, transparency and naming. We defined new methods, headers and message flow as follows.

- SIP methods.
  - Find.

    Used to locate context-related entities in given locations, or services with given capabilities.

  - Provide.

    Used as a response to Find to provide context-based information.

  - Associate.

    Used as a push technique to bring entities together according to context. For instance, it can be used to relate a printer to a user by location.

- SIP headers.
  - ContextType.

An attribute used by the methods above to define the context parameters in the SIP message, and represented by the ontology that has been discussed.

The CSCP involves four phases:

Agent registration and subscription phase: The CMA, AMS and DF register with the SIP proxy server in order to achieve platform discovery in inter-domain communication.

The OA and the CPAs register with the CMA for discovery, lookup and SIP parameters such as SIP addressing in intra-domain communication. The agent's platform could be combined either with the SIP proxy, or in another moderately computing-rich node. For reasons of scalability and performance, CPAs do not engage with the platform in any other situation except inter-domain communication setup. Communication between the user agents and the CPAs are peer-to-peer, as we shall see in the following sections.

Context-based session setup: To establish a session, the user agent sends a SIP invitation message to the domain's CMA (Fig. 6), indicating that the subject is to set up a session by connecting with the agent platform. This facilitates the look-up and discovery process of agents in the system. The SIP server locates the CMA. When the CMA accepts the invitation, the UA and the CMA start a negotiation phase by proposing the context and the ontology that the domain supports. When the set-up is successful, the CMA engages the context providers that can meet the required specifications.

Context invocation within the same domain: During the session, the user may issue a context invocation using the *Find* method in a SIP message with the header indicating the type of context required. The SIP message is directed to any of the context providers assigned to the user during the setup. The receiving CPA will validate the message context; if the request is valid, the CPAs search for the CPA that can provide the required context. The provider is found by searching look-up tables. Upon receiving the *find* message, the CPA will send a message with the *provide* method that contains the context required. In addition, during the session, the CPAs are continuously associating context-based services and information to the user.

Context invocation across different domains: Users may request contextual information from remote domains. This is no trivial task as it requires a discovery process, negotiation

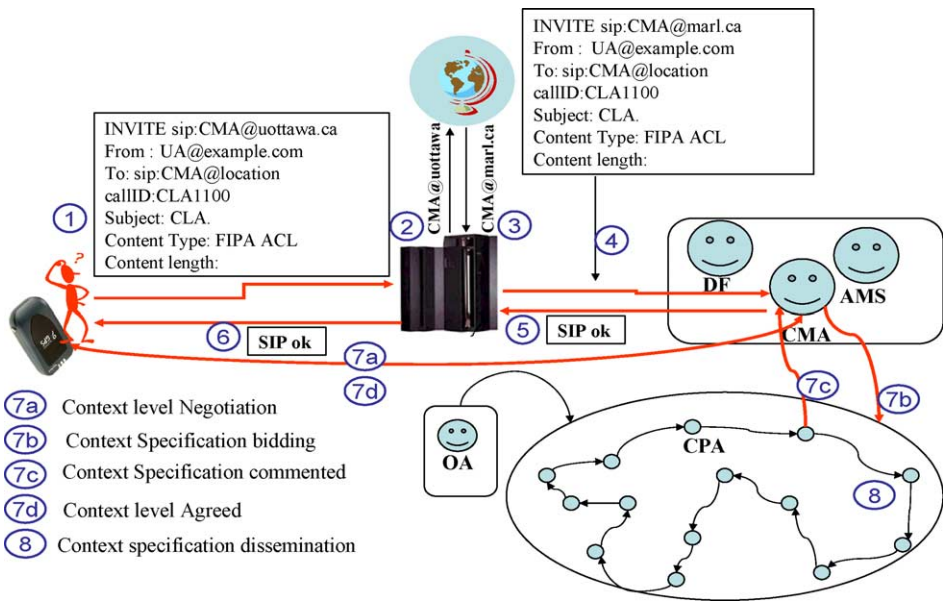

Fig. 6. Context based session setup diagram.

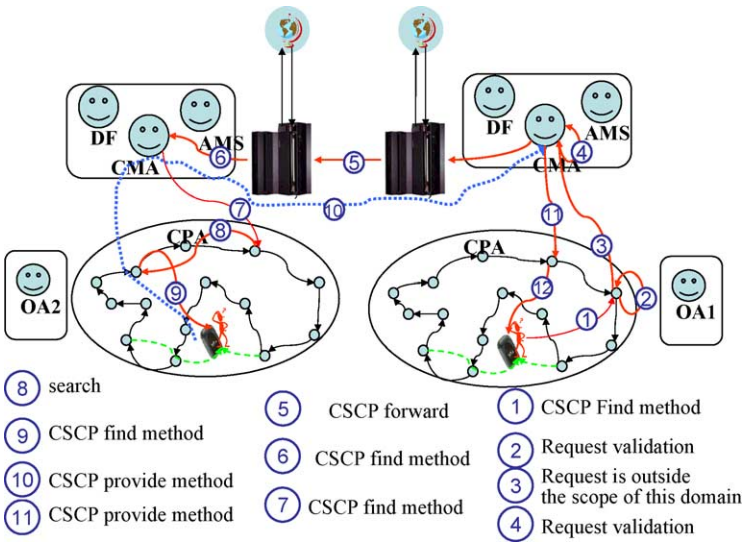

Fig. 7. Inter-domain context invocation.

between different domains, and the need to define context-level agreements in a transparent way between domains. The innovative technique of registering the FIPA compliant mandatory agents, the AMS and the DF, with the SIP proxy has facilitated the process of inter-domain platform discovery and provided the mechanism for cross-domain context invocation. The result is that context-based services in remote domains are invoked and associated to currently available users.

A user in one domain can send a SIP *Find* method to any CPAs requesting context information available in a remote domain, such as weather information. As shown in Fig. 7, the CPAs validate the request and forward it to the CMA as they found that the required context is out of the scope of this domain. The CMA will then consult the location server for the SIP address of the remote CMA. The CMA next forwards the request message to the remote CMA using its SIP address. The remote CMA issues a SIP *Find* method to available CPAs. The CPAs start a search process similar to the one discussed in the intra-domain context invocation. When the required context information is located, the remote CPAs send a SIP *provide* method to the CMA. The CMA sends back the information to the original domain, and an inter-domain communication is set up between the two domains.

6. ACAI implementation and evaluation

To evaluate ACAI, we designed and implemented a prototype system as proof of concept to test applicability, scalability and context effectiveness in everyday activities. The prototype includes three domains with two categories of location. The first category is the propinquity domain represented by the Multimedia and Mobile Agent Research Laboratory (MMARL) at the University of Ottawa, and the National Research Council of

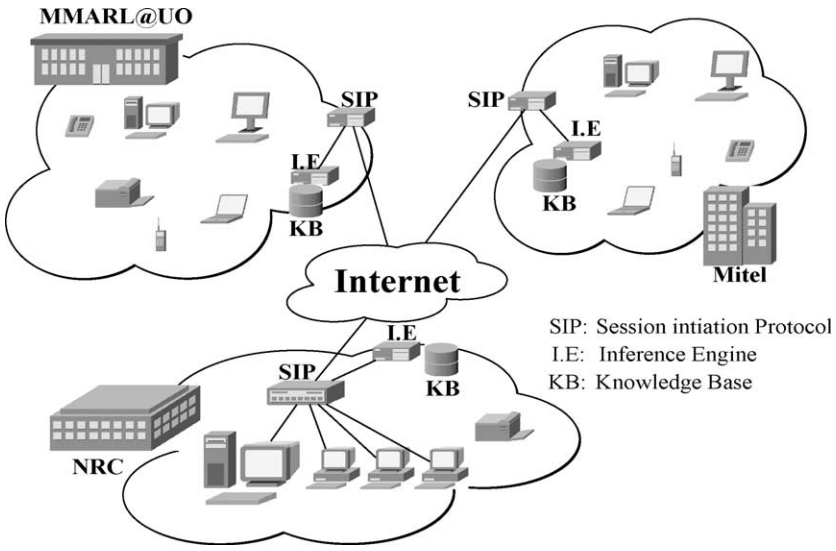

Fig. 8. ACAI prototype in three different domains and the components implemented between the domains to provide spontaneous group communications.

Canada (NRC). The second category is the remote domain represented by Mitel Corporation, as shown in Fig. 8.

Each domain is wirelessly connected using IEEE 802.11 WLAN as well as a Bluetooth connection at NRC. We use the RF-Tag system to provide identity and location as context, network agents for network context information, and service agents for projecting the services in the environment. The context ontology is built using the protege2000 ontology editor[1] and generated in OWL format. The reasoning and inference engine are built using the Jena toolkit from HP. The context-sensitive communication protocol is currently implemented using an extended version of IBM SIP.

In contrast to Gaia (Román et al., 2002) and RCSM (Yau et al., 2002), ACAI is a multi-dimension active space engaged and integrated to provide a seamless, spontaneous environment. We implemented a conferencing application as follows. Tom (a Mitel employee) has a meeting with Karmouch (a UO professor) to discuss some future trends and the applicability of ACAI. Tom is equipped with an RF-Tag, laptop and a PDA that is running a lightweight leap agent platform and his personal agent. Karmouch's personal agent knows that Tom has a meeting with Karmouch. It informs the UO agent that Tom is expected to be at the university as a visitor this afternoon and asks the UO SIP server to register Tom's SIP address. When Tom arrives, the RF reader detects his presence through the PR-tag, and maps the tag to Tom's registered SIP address. The inference engine consequently triggers the network and social dependency predicates. The first rule provides Tom's devices with network preferences, while the second informs Karmouch of Tom presence. During the meeting, Tom suggests that Roger (NRC) and Khedr join the meeting. Tom's personal agent invokes the web server at MMARL and connects to Tom's

[1] http://protege.stanford.edu.

PC at Mitel in order to retrieve Roger's SIP address. It then places a SIP call to Roger. Roger accepts the call, his presence is projected to users in the meeting at MMARL and the meeting agent invokes the camera and speakers to set up the videoconference. Since Khedr is the principal student in the project, the inference engine uses Khedr's profile, shown in Fig. 9, and the location information to locate him, and invite him to join the meeting.

Fig. 10 shows the performance of the system with high and medium rates of service requests. The top graphs give the average detection time, and show how quickly the CPAs detected the service request and how quickly the CMA processed them. As the number of concurrent service requests increased, the average detection time increased at a rate that was higher than the CPA response time. For a total of 120 requests, it took an average of nearly 14 s to detect a request. The maximum wait was a slow 45 s. With a medium request rate, the 120 requests took an average detection time of 1.7 s with a maximum delay of 9 s only. The lower graphs show that average waiting time depends on two factors: the average service time, and the availability of the service. We first conducted the experiment considering location as the only context. We then repeated the experiment taking other context information into account, such as the current resources and agent interaction. With an average completion time of 10 s, performance was found to be much better in the second test than the first. This is because more information is available to define the context. The second experiment examined the increase in the average service-completion

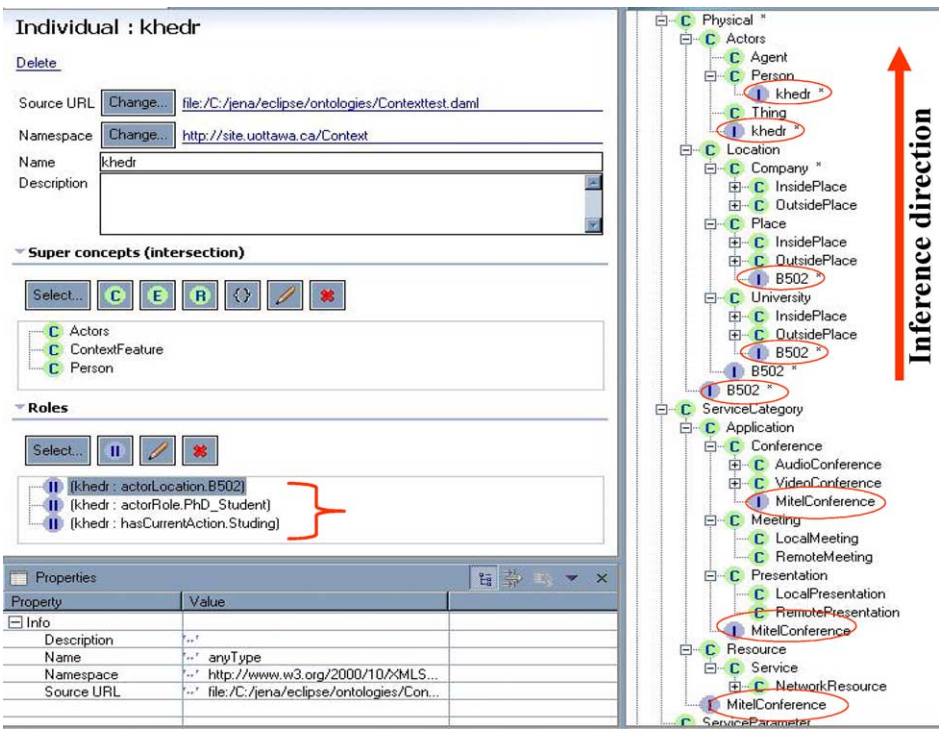

Fig. 9. The user profile of Khedr and location.

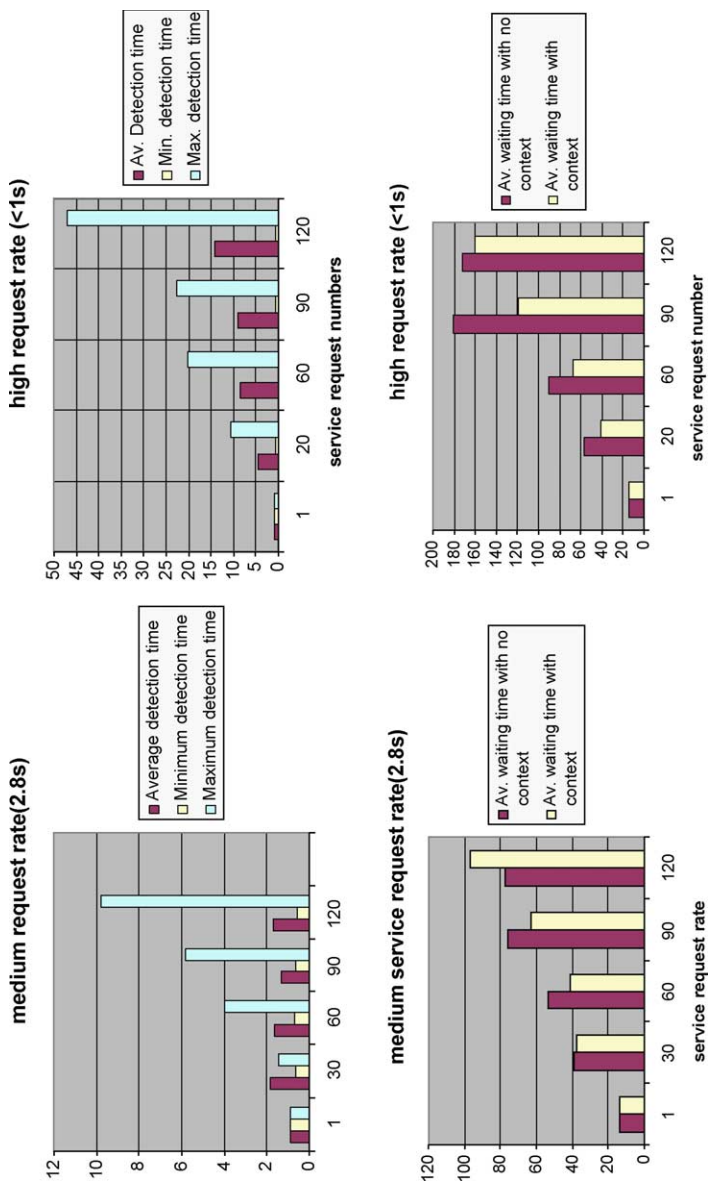

Fig. 10. Experiment results.

Table 1
Service time vs utilization, availability and % of success

Service time	Group name	Average utilization		Average availability		Standard deviation		% of Service replied	
		Without context	With context	Without context	With context	Without context	With context	Without context	With context
10	Printer (6)	0.174	0.174	5.750	5.750	0.4	0.4	100	100
	INT (4)	0.36	0.36	3.59	3.59	0.529	0.529		
	Proj (3)	0.311	0.311	2.687	2.686	0.49	0.49		
	Email (1)	0.309	0.309	0.689	0.689	0.46	0.46		
	Phone (1)	0.2	0.2	0.792	0.792	0.4	0.4		
30	Printer (6)	0.636	0.669	5.317	5.284	0.758	0.723	93.7	94.9
	INT (4)	0.97	1.019	2.989	2.939	0.934	0.92		
	Proj (3)	0.894	0.385	2.101	2.143	0.714	0.839		
	Email (1)	0.936	0.917	0.061	0.08	0.244	0.275		
	Phone (1)	0.211	0.299	0.787	0.698	0.408	0.458		
50	Printer (6)	0.587	0.684	5.344	5.257	0.684	0.762	78.1	86.1
	INT (4)	1.425	1.654	2.527	2.301	0.718	1.195		
	Proj (3)	1.689	1.806	1.308	1.189	0.767	1.04		
	Email (1)	0.974	0.917	0.024	0.081	0.16	0.275		
	Phone (1)	0.735	0.876	0.262	0.115	0.411	0.33		
90	Printer (6)	2.171	2.06	3.792	3.869	1.401	1.457	60	72
	INT (4)	2.911	2.86	1.056	1.1	1.358	1.311		
	Proj (3)	1.642	2.844	1.355	0.151	0.57	0.617		
	Email (1)	0.963	0.971	0.037	0.024	0.190	0.168		
	Phone (1)	0.873	0.921	0.118	0.076	0.333	0.27		

time while keeping the number of requests the same. We measured the average utilization, and the percentage of service available per user request. Again, we considered two cases: first using location as the only context information and then adding other context parameters such as shortest busy, longest idle, and agent interaction.

Table 1 shows that, for low service time, using context has no apparent effect, with results the same in both situations. But as we increased the service invocation time, the average utilization and average availability of the services with no context were worse than when context interaction was used. The overall percentage of service access deteriorated at a higher rate. It can also be seen that the use of context and context interaction tries to balance the use of services between requests as the service time increases. This makes the system more stable and robust.

7. Related work

Little current research in the area of context-awareness explicitly focuses on designing an infrastructure for spontaneous applications that is suited for widely deployed pervasive

computing environments. No other proposals explicitly deal with context as an ontological model in which context information is separated from its processing. Most of the current work is concerned with providing frameworks and toolkits that support context representation through conventional methods which limit interoperability and ease of integration.

The Gaia project (Román et al., 2002) defines a metaoperating system that is used to manage ubiquitous computing habitats and active spaces. Gaia recognizes the need for context information in what is called the context service component. The context service lets applications query and register for particular context information, which helps them adapt to their environment. A registry maintains a list of available context providers and applications can use the registry to find the providers of the context they desire. They use a first order logic for modeling context, which allows them to write rules that describe actions. However, Gaia does not support a peer discovery mechanism between the applications and the context providers. Applications can only use the context providers available in the registry. We argue that this will introduce scalability issues when the number of application instances increases. It will not handle situations like temporary failure or unavailability due to overload. In addition, Gaia does not support ontology-based context modeling as proposed in this paper. This will limit the ability of applications using Gaia active space to advertise the kinds of context they are interested in.

The RCSM (Yau et al., 2002) is a middleware that facilitates the development and runtime operation of context-sensitive software. RCSM is based on ORB; it provides what is called the adaptive object container (ADC) for runtime context data acquisition. The RCSM-IDL compiler generates a custom-made ADC tailored for a particular context-sensitive object. The context-sensitive interface lists the contexts the applications use, a list of actions the applications provide, and a mapping between them that clearly indicates, based on specific context values, when an action should be completed. Our infrastructure outperforms the RCSM in two situations. First, its ability to separate context knowledge from context processing allows it to work in situations where context is incomplete or ambiguous. Second, RCSM does not support context inference and composition. RCSM needs to be modified as new context sources with no verification mechanism are added. The different situational actions and the mapping between the actions and the context list needs to be stated, as RCSM has no clear method for composing context in a semantic way, nor to reason even if the context is composed.

Solar (Chen and Kotz, 2003) has a similar objectives to ACAI in that it defines a pervasive computing infrastructure for context-aware mobile computing. Solar is built on the concept of graph-based abstraction for collecting, aggregating and disseminating context information. The abstraction models context as events that flow through a directed acyclic graph of event-processing operators and are delivered to subscribing events.

But as the author mentions in Chen and Kotz (2003), Solar is not designed for ad hoc communications and the consequent spontaneous applications. This is because all the services for the naming, and naming-related discovery process are embedded in the infrastructure. Solar has the capability for resource discovery through the naming of the context source information with context-related information such as location. This contrasts with ACAI's concept where resources are discovered according to

the current context. This avoids the need for context providers to be part of the context information.

8. Conclusion and future work

In this paper, we present a new ACAI for mobile users and spontaneous applications. The infrastructure provides a wide range of functionalities in different layers of its infrastructure. Our model for context representation is both simple and composite, and is used by other modules in the infrastructure. The multi-agent system includes the CMA responsible for managing context interaction between end-user agents and context-provider agents. The proposed context-sensitive communication protocol allows users and context providers to clearly define their aspects of context information and context specifications. We developed a prototype of the multi-agent system using JADE (Vitaglione et al., 2002) and a java API of the context ontology model as a proof of concept. Currently we are working on extending the model with a fuzzy inference capability that will allow agents to reason from uncertain context, and to provide actions that are not explicitly defined. The context management system is also being designed to provide scalable, persistent and consistent context information to end-users and services.

References

Chen G, Kotz D. Context-sensitive resource discovery. First IEEE International Conference on Pervasive Computing and Communications; 2003.

Davies N, Gellersen H-W. Beyond prototypes: challenges in deploying ubiquitous systems. IEEE Pervas Comput 2002.

Dey A. Understanding and using context. Personal Ubi Comput 2001;5(1).

Dey A, Abowd G. Towards a better understanding of context and context-awareness. First International Symposium on Handheld and Ubiquitous Computing (HUC'99), June; 1999.

Harroud H, Ahmed M, Karmouch A. Policy-driven personalized multimedia services for mobile users. IEEE Trans Mobile Comput 2003.

Henricksen, Indulska J, Rakotonirainy A. Generating context management infrastructure from high-level context models. Fourth International Conference on Mobile Data Management, January, Melbourne, Australia; 2003.

Khedr M, Karmouch A. Exploiting SIP and agents for smart context level agreements. 2003 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, August 28–30, Victoria, BC, Canada; 2003.

Khedr M, Karmouch A. A Semantic Approach for Negotiating Context Information in Context Aware Systems. IEEE Intell Syst Mag 2004; (in press).

Khedr M, Karmouch A, Liscano R, Gray T. Agent-based Context Aware Ad hoc Communication, MATA'02, October. 2002.

Lum WY, Lau FCM, Context-Aware A. Decision engine for content adaptation. IEEE Pervas Comput 2002;1(3).

Román M, et al. A middleware infrastructure for active spaces. IEEE Pervas Comput 2002;Oct–Dec:74–83.

Schulzrinne H, Rosenberg J. Session Initiation Protocol (SIP) caller preferences and callee capabilities, Internet Draft, Internet Engineering Task Force, November. 2002.

Vitaglione G, Quarta F, Cortese E. Scalability and performance of jade message transport system. AAMAS Workshop on AgentCities, Bologna, July; 2002.

Yau S, et al. Reconfigurable context-sensitive middleware for pervasive computing. IEEE Pervas Comput 2002; 1(3).

Mohamed Khedr is a PhD candidate at the University of Ottawa, Canada. His research interests include pervasive computing, agent-based middlewares, context-aware environments and semantic information modeling. He received his BSc and MS from Arab Academy for Science and Technology, Egypt.

Ahmed Karmouch received his MS and PhD degrees in computer science from the University of Paul Sabatier, Toulouse, France, in 1976 and 1979 respectively. From 1976 to 1983, he was a Research Engineer at INRIA (Institut National de Recherche en Informatique et en Automatique) Paris, France. First in distributed databases with the Sirius Project, and then with the Kayak Project where he researched Office Information Systems and was responsible for the Multimedia Distributed Message System Research Group.

From 1984 to 1988 he was with Bull SA, Paris, France, as a Senior Manager at the Department of Advanced Studies. He was responsible for the Distributed Multimedia Document Management Group. From 1988 to 1991, he was Director of research on Multimedia Distributed Databases and Architectures at the Ottawa Medical Communications Research Group, University of Ottawa.

Since 1991 Dr. Karmouch has been Professor of Electrical and Computer Engineering and computer Science at the School of Information Technology and Engineering, University of Ottawa. He also holds an Industrial Research Chair from the Ottawa Carleton Research Institute and Natural Sciences and Engineering Research Council. He has been Director of the Ottawa Carleton Institute for Electrical and Computer Engineering. Dr. Karmouch is involved in several projects with Telecommunications Research Institute of Ontario, Nortel Networks, Bell Canada, Mitel, National Research Council Canada, Centre National de Recherche Scientique in France, March Networks, CANARIE, Communications & Information Technology Ontario (Cito), and TeleLearning National Center of Excellence. Dr. Karmouch is a partner in the "WWI: Ambient Networks", a European Sixth Framework Integrated Project. His current 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. Dr. Karmouch has published over 160 papers in the area of multimedia systems, ad hoc communications and mobile agents for Telecommunications, he is member of ACM and IEEE, he has served in several program committees, organized several conferences and workshops and served as Guest Editor for IEEE Communications magazine, Computer Communications, MTAP, and others.